The Dissertation Committee for Song Han

certifies that this is the approved version of the following dissertation:

Networking Infrastructure and Data Management for Large-Scale Cyber-Physical Systems

Committee:

Aloysius K. Mok, Supervisor

Simon S. Lam

Farnam Jahanian

Tei-Wei Kuo

Lili Qiu

Yin Zhang

Networking Infrastructure and Data Management for Large-Scale Cyber-Physical Systems

by

Song Han, B.S.; M.A.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2012

Dedicated to my family for their love and support

Acknowledgments

I would like to express my deepest gratitude to Professor Aloysius K. Mok, my thesis advisor, for his excellent guidance and great support throughout my Ph.D study in the University of Texas at Austin. Without his encouragement and help, this thesis would not have been possible.

I would like to thank Mr. Mark Nixon, who introduced me into the field of wireless industrial process control, helped me develop my background in wireless communication and advanced control, and financially supports my research over the years.

I would like to thank my committee members, Dr. Simon S. Lam, Dr. Farnam Jahanian, Dr. Tei-wei Kuo, Dr. Lili Qiu and Dr. Yin Zhang for their invaluable suggestions and detailed comments on my thesis. I would like to thank Dr. Krithi Ramamritham, Dr. Kam-yiu Lam, Dr. Edward Chan, Dr. Reynold Cheng, Dr. Sang H. Son, Mr. Jiantao Wang, Dr. Terry Blevins, Dr. Deji Chen, Mr. Mike Lucas, Dr. Ming Xiong, Dr. Luis Sentis, Mr. Kwan Suk Kim, Dr. Risto Miikkulainen, Dr. Lawrence Waugh, Mr. Fred Stotz, Dr. Tianji Li, Dr. Douglas Leith, Mr. Chen Qian, Dr. Masayoshi Tomizuka, Dr. Nancy Byl, Mr. Wenlong Zhang, Mr. Wally Pratt, Ms. Veena Gondhalekar, Dr. Junghoon Lee, Dr. Guihai Chen, Mr. Zifei Zhong and Mr. Hongxing Li who I have the honor to collaborate with during the past six years, for their inspiration on research ideas and constructive comments on technical details. I would like to thank people from my research group, Pak Ho Chung, Pei-Chi Huang, Quan Leng, Zheng Li, Pei-Shu Liao, Jianyong Meng, Wing-Chi Poon, Jianping Song, Jianliang Yi, Yi-Hung Wei and Xiuming Zhu for their fruitful discussions

and great support.

I would also like to acknowledge my friends and colleagues for all memorable moments in Austin: Dongliang Xu, Zheng Lou, Yu Tao, Yang Wang, Xu Wang, Hang Yu, Zuocheng Ren, Hongkun Yang, Yu Li, Na Meng, Dong Li, Xin Sui, Wei Tang, Qiang Zhang, Jiandan Zhen, Shaohua Guo, Yibo Jiao, etc.

Finally, my deepest appreciation to my wife Feng Zou, my mother Keke Song and my father Changzao Han, for their endless love, support and understanding.

SONG HAN

The University of Texas at Austin
December 2012

Networking Infrastructure and Data Management for Large-Scale Cyber-Physical Systems

Publication No. _____

Song Han, Ph.D.
The University of Texas at Austin, 2012

Supervisor: Aloysius K. Mok

A cyber-physical system (CPS) is a system featuring a tight combination of, and coordination between, the system's computational and physical elements. A large-scale CPS usually consists of several subsystems which are formed by networked sensors and actuators, and deployed in different locations. These subsystems interact with the physical world and execute specific monitoring and control functions. How to organize the sensors and actuators inside each subsystem and interconnect these physically separated subsystems together to achieve secure, reliable and real-time communication is a big challenge. In this thesis, we first present a TDMA-based low-power and secure real-time wireless protocol. This protocol can serve as an ideal communication infrastructure for CPS subsystems which require flexible topology control, secure and reliable communication and adjustable real-time

service support. We then describe the network management techniques designed for ensuring the reliable routing and real-time services inside the subsystems and data management techniques for maintaining the quality of the sampled data from the physical world. To evaluate these proposed techniques, we built a prototype system and deployed it in different environments for performance measurement. We also present a light-weighted and scalable solution for interconnecting heterogeneous CPS subsystems together through a slim IP adaptation layer and a constrained application protocol layer. This approach makes the underlying connectivity technologies transparent to the application developers thus enables rapid application development and efficient migration among different CPS platforms. At the end of this thesis, we present a semi-autonomous robotic system called cyberphysical avatar. The cyberphysical avatar is built based on our proposed network infrastructure and data management techniques. By integrating recent advance in body-compliant control in robotics, and neuroevolution in machine learning, the cyberphysical avatar can adjust to an unstructured environment and perform physical tasks subject to critical timing constraints while under human supervision.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cyber-physical systems (CPS) consist of the class of large-scale infrastructures that have significant cyber and physical components and have wide-ranging impact on society in their deployment. Because of the proliferation of low-cost and increased-capability sensors, the rapid advancement in low-power, high-capacity computing devices, the revolution of wireless communication technologies, and continuing improvements in energy capacity, the research in cyber-physical systems has received tremendous interests in recent years [67, 50, 76, 20, 39]. Examples of CPS include energy management systems, water resource monitoring and control, logistics and disaster management, eldercare systems, intelligent robots and smart structural systems. A common feature of cyber-physical systems is that they are deeply embedded and interact with the physical world, and must operate dependably, safely, securely and support a wide range of real-time services.

Figure 1.1 depicts the typical architecture of a large-scale cyber-physical system. A large-scale CPS usually consists of several heterogeneous subsystems designed and engineered to achieve specific monitoring and control functions. In such CPS, each subsystem is typically a networked embedded system based on a certain wired or wireless connectivity technology, and these subsystems are usually deployed in different locations and connected through the Internet. The network infrastructure of large-scale CPS should provide an

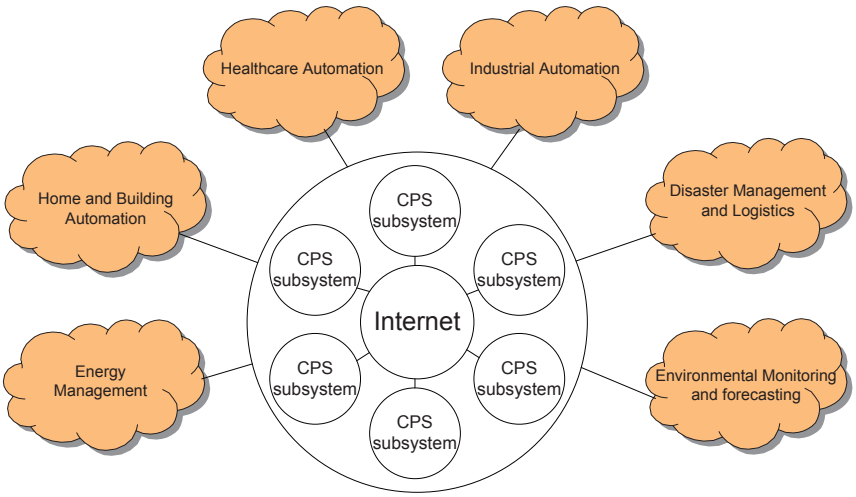

Figure 1.1: A typical architecture of large-scale networked cyber-physical systems

abstraction so that CPS applications built on top of it are transparent to the connectivity technologies adopted in each individual CPS subsystem. A typical example of CPS is our ongoing cyber-physical avatar project which aims at building a semi-autonomous robot system that can adjust to an unstructured environment and perform physical tasks subject to critical timing constraints while under human supervision. A cyberphysical avatar is semi-autonomous in that there are actions it must take without human intervention because of the relatively short timing constraints, e.g., the control loop that maintains a fast walking gait. On the other hand, a cyberphysical avatar should not be programmed to deal with only a fixed set of scenarios because we cannot foresee all the contingencies in all operational environments, e.g., a building on fire in a rescue mission. An effective interface between the cyberphysical avatar and its human supervisor is essential for success, and this requires the cyberphysical avatar to be designed for predictable and timely response. As the cyber-physical avatar gains more physical skills, it can be trusted to perform more subtasks on its own.

There are many design challenges to be addressed in building a cyber-physical system like the cyberphysical avatar. These challenges are summarized below. This thesis

aims at addressing these challenges and providing a foundation for designing networking infrastructure and data management techniques to help build large-scale reliable, secure and adjustable cyber-physical systems supporting a wide range of time-critical services.

- **Achieving reliable and real-time services in CPS subsystems:** A typical CPS subsystem consists of sensors and actuators which are attached to physical entities. These sensors and actuators are usually networked together to provide secure, reliable and time-critical monitoring and control services. Although extensive research has already been done on sensor networks in the literature, most of them failed to satisfy all the requirements especially for the deterministic real-time performance guarantee. Designing a secure and reliable real-time communication protocol especially in wireless environment to serve as the general platform for CPS subsystems is desired by many cyber-physical systems.

- **Interconnecting CPS subsystems:** CPS subsystems could be deployed in different locations, and collaborate with each other to achieve mission-critical tasks. Due the limitation of the existing Internet infrastructure which cannot provide any *QoS* guarantee on the end-to-end packet delivery, a new programmable network infrastructure with programmable switches should be applied. On the other hand, interconnecting heterogeneous CPS subsystems requires enhancement on sensors and actuators in each subsystem with IP functions. However, embedded devices usually have very limited memory, power, computing and communication capability. Due to these resource constraints, it is not realistic to equip them with a complete TCP/IP stack. Instead, a simple but sufficient IP adaptation layer should be designed and applied to hide the difference of heterogeneous physical and data link layer specifications in the subsystems and provide intra- and inter-subsystem IP service.

- **Maintaining data freshness and control quality in cyber-physical systems:** A CPS is usually deeply embedded in the physical world to collect dynamic physical

measurements which will be periodically published to a real-time database either actively or passively. Queries from other subsystems in the CPS will be performed on these data and the results will be used to execute necessary control actions. To make sure that the control tasks are executed properly, it is critical to guarantee the freshness of the data maintained in the CPS subsystems. Efficient algorithms should be designed to decide how to execute and schedule the sensor update and control tasks together to maintain the data freshness and control quality while minimizing the system and network overhead. This requirement becomes more stringent when the CPS exhibits a multi-modal behavior and the update and control workload in the system is dynamic.

In the first part of this thesis, we introduce a wireless real-time communication protocol called WirelessHART which can serve as a good candidate for the network infrastructure of CPS subsystems to support secure, reliable and real-time services. WirelessHART is a TDMA-based secure and mesh networking technology. The TDMA-based communication infrastructure and the network-wide synchronization make it possible to allocate communication resource between the peers in a deterministic way so that end-to-end real-time requirements can be satisfied. The built-in packet retransmission, packet-level channel hopping and channel blacklist techniques in the data link layer and the reliable graph routing approaches in the network layer will help the system achieve high level reliability. To achieve secure communication, the MAC layer provides hop-to-hop data integrity by using MIC and the network layer employs various keys to provide confidentiality and data integrity for end-to-end connections. We present the typical topology of WirelessHART networks and describe the architecture and detailed design of the communication protocol in Chapter 2. According to different communication purposes, in Chapter 3, we define three types of reliable routing graphs in WirelessHART networks and present efficient algorithms to construct them and describe the recovery mechanisms. Based on these routing graphs, we further describe how to construct the data link layer communication schedule in the network

to achieve end-to-end real-time performance. The design, implementation, deployment and measurement of our prototype system is presented in Chapter 4.

In the second part of this thesis, we present our IP-based solution in Chapter 5 to organize and interconnect CPS subsystems into a large-scale cyber-physical system for supporting a wide range of monitoring and control services. Instead of applying a complete TCP/IP protocol in resource-constrained embedded sensors and actuators which is usually inapplicable, we propose to design a slim IP adaptation layer on the Gateway of each CPS subsystem and on the embedded devices which need to support IP services. Under this design, the CPS subsystems adopting different connectivity technologies will keep their network formation, intra-network routing and the security mechanisms unchanged. IP traffic between peers located in different CPS subsystems will be compressed, fragmented and wrapped as normal packets in the corresponding subsystems. The Gateways of the involved subsystems will execute the adaptation functions to resume the original IP packets and transmit them in the backbone network. In this infrastructure, normal traffic inside each CPS subsystem and IP traffic among different subsystems can co-exist simultaneously and incremental deployment is well supported. To achieve deterministic performance in end-to-end communication among CPS subsystems, we are deploying programmable switches to interconnect heterogeneous CPS subsystems together to achieve remote, reliable and real-time services.

In the third part of this thesis, we study the data management issues in CPS subsystems where sensor measurements need to be sampled from the physical world and updated to a real-time database (RTDB) for the execution of queries and control tasks from itself or other subsystems. To guarantee the qualities of the query results and control performance, senor data values in the RTDB should be always maintained fresh. In Chapter 6 we propose a series of algorithms aiming at maintaining the sensor data freshness while minimizing the workload imposed by the sensor updates. We first propose a novel algorithm for fixed-priority systems called deferrable scheduling (*DS-FP*) in Section 6.2. *DS-FP* is based

on sporadic task model and is proved to outperform the periodic solutions in providing better schedulability and lower system workload. A necessary and sufficient schedulability condition and overhead reduction techniques are also proposed to further improve its performance. To maintain data freshness in systems requiring dynamic scheduling, we further propose a dynamic scheduling algorithm called deferrable scheduling with least actual laxity first (*DS-LALF*) in Section 6.5.2, and present its schedulability analysis. Chapter 7 studies the problem how to maintain the freshness of real-time data in the presence of mode changes in dynamic CPS subsystems. We proposed to use different scheduling policies in different modes and introduced two algorithms to search for proper switch points. These two algorithms make sure that data freshness is maintained not only in every mode of the dynamic system but also during the mode change. Aforementioned algorithms are effective in maintaining real-time data freshness, but they have ignored the impact on the performance of the control tasks which are mostly executed concurrently with the update transactions. Obviously, the co-scheduling of these two types of tasks conflict with each other as both of them need to meet the deadlines and at the same time compete for the same set of resources for processing. To address this problem, in Chapter 8, we extend *DS-LALF* to be a dynamic co-scheduling algorithm, called *Co-LALF*. The performance goal of *Co-LALF* is to construct a schedule in RTDB that can meet the deadlines of all the periodic control transactions and at the same time maximize the quality of data (QoD) of the data objects for execution of the control transactions.

The resulting networking infrastructure and data management techniques in this thesis for large-scale cyber-physical systems will provide a general framework for a wide range of CPS applications. The proposed real-time wireless communication protocol along with its network and data management techniques can serve as an ideal candidate for CPS subsystems which require secure and reliable communication, low power consumption and real-time service support. The proposed programmable networks plus IP-based solution for interconnecting subsystems provide a light-weight and scalable approach to integrating

physically separated heterogeneous CPS subsystems together. Under this infrastructure, cyber-physical systems and the applications built on top of them will be able to support incremental deployment without affecting the existing services.

In the final part of this thesis, we briefly describe several CPS applications that we are building, and present a futuristic application called cyberphysical avatar in Chapter 9. Cyber-physical avatar is defined to be a semi-autonomous robotic system that adjusts to an unstructured environment and performs physical tasks subject to critical timing constraints while under human supervision. Cyberphysical avatar is built based on our proposed network infrastructure and data management techniques. It also integrates the recent advance in another two technologies: body-compliant control in robotics, and neuroevolution in machine learning. Body-compliant control is essential for operator safety since cyberphysical avatars perform cooperative tasks in close proximity to humans. Neuroevolution technique is essential for programming cyberphysical avatars inasmuch as they are to be used by non-experts for a large array of tasks, some unforeseen, in an unstructured environment. By integrating all these technologies, we have built a prototype cyberphysical avatar testbed to validate our design in this thesis.

Chapter 2

Real-Time Wireless Protocol for Cyber-Physical Systems

A CPS subsystem usually comprises a network of physically distributed embedded sensors and actuators which are battery-powered and equipped with constrained computation resources and limited communicating capabilities. Since most of these subsystems are designed for critical sensing and control applications and are usually deployed in harsh and noisy environments, they require secure, reliable and real-time reactions. In this chapter, we present the design and implementation of a TDMA-based secure wireless communication protocol called WirelessHART which is specifically designed for wireless sensing and control networks. WirelessHART serves as a good candidate for the network infrastructure of CPS subsystems to support secure, reliable and real-time services while taking energy saving into consideration as well. We present the background knowledge of WirelessHART network and its comparison with several other publicly available wireless standards in Section 2.1. Section 2.2 to Section 2.7 elaborate the architecture of WirelessHART protocol and describe the challenges we met during the implementation and our corresponding solutions. We conclude this chapter in Section 2.8.

2.1 Background

WirelessHART is a secure and energy-efficient wireless communication protocol for supporting end-to-end real-time performance in mission-critical wireless sensing and control applications. Figure 2.1 illustrates the architecture of the WirelessHART protocol according to the OSI 7-layer communication model. At the bottom of its communication stack, WirelessHART adopts IEEE 802.15.4-2006 [9] physical layer for supporting low-power wireless communication. On top of that, WirelessHART has its own time-synchronized data link layer. Some notable features of WirelessHART data link layer include strict 10 ms timeslot, network-wide time synchronization, channel hopping, channel blacklisting, and industry-standard AES-128 ciphers and keys. The network layer supports self-organizing and self-healing mesh networking techniques and uses source routing and graph routing. In this way, messages can be routed around interferences and obstacles and greatly improve the network performance in noisy and harsh environments. WirelessHART distinguishes itself from other public standards by maintaining a central Network Manager. The Network Manager is responsible for maintaining up-to-date routes and communication schedules for the network, thus guaranteeing the reliable and real-time network communications.

Figure 2.2 shows a typical topology of a WirelessHART mesh network. All WirelessHART nodes support the basic mesh node functionalities, including routing capability. The basic node types of a WirelessHART network are: 1) **Network Manager** which is responsible for configuring the network, scheduling and managing communication among WirelessHART devices. It is implemented in software that resides in the Gateway or the Host; 2) **Gateway** which connects Host applications with field devices. It is responsible for data caching and query processing; 3) **Access Point** which is attached to the Gateway and provides redundant paths between the wireless network and the Gateway; 4) **Router** which is deployed in the network to improve network coverage and connectivity; 5) **Field Device** which is attached to the process plant and could be a sensor or an actuator; 6) **Handheld** which is a portable WirelessHART-enabled computer used to configure devices, run diag-

nostics, and perform calibrations; and 7) **Adapter** which is a bridge device between the wireless mesh network and traditional wired HART devices.

Besides WirelessHART, there are a few publicly available wireless standards on office and manufacturing automation, such as ZigBee [13], Bluetooth [2] and Wi-Fi [8]. However, these technologies cannot meet the stringent requirements of time-critical CPS applications. Compared with office applications, CPS applications have stricter timing requirement and higher security concern. None of them makes any effort to provide a guarantee on end-to-end wireless communication delay. In addition, environments where CPS will be deployed are harsher for wireless applications in terms of interferences and obstacles than office environment. Some interferences may be persistent.

Both WirelessHART and Bluetooth support time slots and channel hopping. However, Bluetooth is targeted at Personal Area Networks (PAN), whose range is usually set to 10 meters. Furthermore, Bluetooth only supports star-type network topology, and one master can only have up to 7 slaves. These limitations make it awkward to apply Bluetooth in large-scale sensing and control systems. In contrast, WirelessHART supports mesh networking directly. The topology of a WirelessHART network can be a star, a cluster or a mesh, thus providing much better scalability. Both WirelessHART and ZigBee are based on the IEEE 802.15.4 physical layer. While ZigBee uses the existing IEEE 802.15.4 MAC, WirelessHART goes one step further to define its own MAC protocol. WirelessHART introduces channel hopping and channel blacklisting into the MAC layer, while ZigBee can only utilize Direct Sequence Spread Spectrum (DSSS) built in IEEE 802.15.4. Thus, if a noise is persistent, which is not unusual in CPS environments, the performance of a ZigBee network might degrade severely. By changing the communication channel pseudo-randomly, WirelessHART can limit the damage to minimum. Wi-Fi is based on the IEEE 802.11 standards and its spectrum assignments and operational limitations are not consistent worldwide. In addition, its power consumption is fairly high compared to other low-bandwidth standards, which makes it not a good fit for CPS environments as well.

OSI Layer	Function	WirelessHART
Application	Provide the User with Network Capable Applications	Command Oriented. Predefined Data Types and Application Procedures
Presentation	Convert Application Data between Network and Local Machine Formats	
Session	Connection Management Services for Applications	
Transport	Provide Network Independent, Transparent Message Transfer	Auto-Segmented transfer of large data sets, reliable stream transport, Negotiated Segment sizes
Network	End to End Routing of Packets. Resolving Network Addresses.	Power-Optimized Redundant Path, Mesh to the edge Network
Data Link	Establish Packet Structure, Framing, Error Detection, Bus Arbitration.	Secure, Time Synched TDMA/CSMA, Frequency Agile
Physical	Mechanical, Electronic Connections. Transmit Raw Bit Stream.	2.4 GHz Wireless, 802.15.4 based radios, 10dBm Tx Power

Figure 2.1: The architecture of WirelessHART protocol

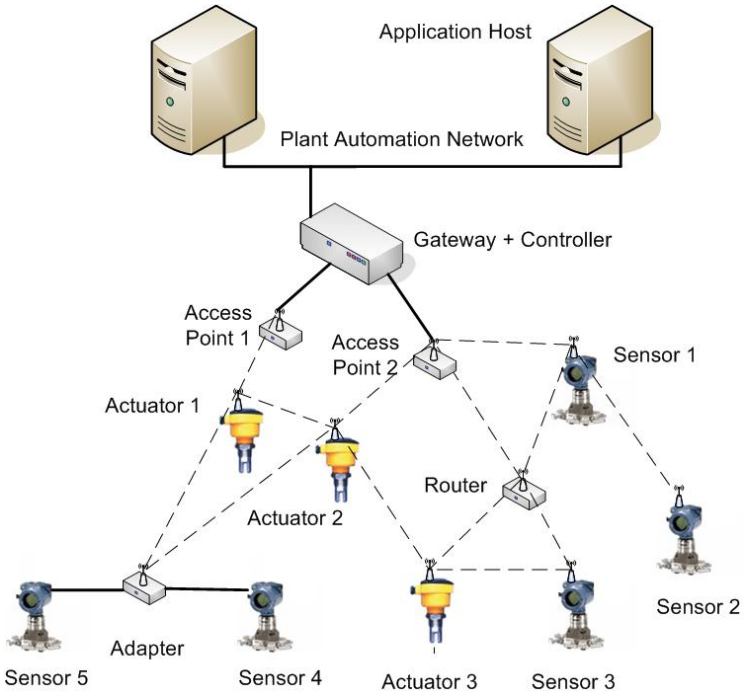

Figure 2.2: A typical topology of WirelessHART network

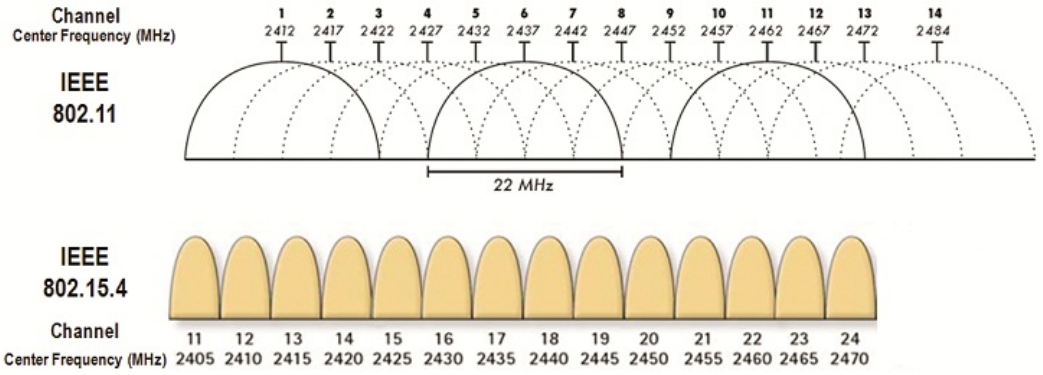

Figure 2.3: 802.11 and 802.15.4 channels in the 2.4 GHz spectrum

2.2 802.15.4-based Physical Layer

The WirelessHART physical layer is based mostly on the IEEE STD 802.15.4-2006 2.4 GHz DSSS physical layer [9]. This layer defines radio characteristics, such as the signaling method, signal strength, and device sensitivity.

Conforming to IEEE 802.15.4 standard [9], WirelessHART operates in the 2400-2483.5 MHz license-free ISM band with a data rate of up to 250 kbits/s. Its channels are numbered from 11 to 26, with a 5 MHz gap between two adjacent channels. Figure 2.3 summarizes 802.15.4 channels in the 2.4 GHz spectrum compared with that of 802.11.

To meet strict timing requirements, WirelessHART mandates that radio transceivers be compliant to the IEEE 802.15.4 standard and meet the following criteria: (1) The maximum switching time between channels shall be 12 symbol periods (0.192 ms); (2) The maximum radio turn-on time should be 4 ms; (3) The power level of the device must be controlled (programmable) at discrete, monotonic levels from -10 dBm to $+10$ dBm EIRP (with ± 2 dBm of error in actual power level); and (4) Receiver sensitivity should be -90 dBm or better.

Term	Definition
AES	Advanced Encryption Standard
ASN	Absolute Slot Number
BDM	Background Debug Module
CCA	Clear Channel Assessment
CCM*	Counter with CBC-MAC (corrected)
DLPDU	Data Link Protocol Data Unit
DSSS	Direct Sequence Spread Spectrum
EIRP	Equivalent Isotropic Radiated Power
MCU	Micro Control Unit
MIC	Message Integrity Code
NPDU	Network Protocol Data Unit
PIB	Protocol Information Base
SP	Service Primitive
TDMA	Time Division Multiple Access

Table 2.1: Definitions and abbreviations

2.3 TDMA-based Data Link Layer

One distinct feature of WirelessHART is the time-synchronized data link layer. WirelessHART defines a strict 10ms time slot and utilizes TDMA technology to provide collision free and deterministic communication. The concepts of superframe and link are used to define the communication behavior among devices; the built-in packet-level channel hopping and channel blacklisting mechanisms are adopted to explore the channel diversity and improve the reliability and promptness of the communication; synchronization mechanisms are applied to achieve network-wide synchronization. In the following of this section, we will first describe the aforementioned features of the data link layer and then present our design details.

2.3.1 Superframe and Link

The concept of *superframe* is introduced to group a sequence of consecutive time slots. Note a superframe is periodical, with the total length of member slots as the period. All

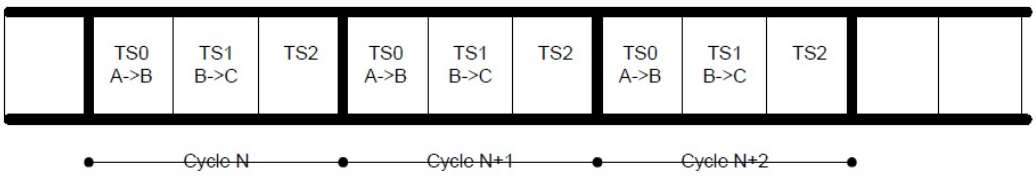

Figure 2.4: Example of a three-slot superframe

superframes in a WirelessHART network start from the same ASN (absolution slot number) 0, the time when the network is first created. Each superframe then repeats itself along the time based on its period. Figure 2.4 shows an example of a three-slot superframe. If we have two superframes A and B, A has 3 slots as shown in Figure 2.4 and B has 4 slots. Then ASN 9 is the first slot in the 4th instance of superframe A, and the 2nd slot in the 3rd instance of superframe B. A WirelessHART device can support multiple superframes so that it can follow different communication schedules upon the request of the Network Manager.

Each time slot in a superframe, if not idle, will have an associated link information. A *link* is described by a vector: <type, source, destination, channel> where *type* indicates the type of the slot (transmit/receive); *source* and *destination* are the addresses of the source device and destination device for the communication on this time slot respectively; and *channel* specifies the logical channel to be used in the communication.

2.3.2 Network-wide Time Synchronization

Our data link layer is based on Time Division Multiple Access (TDMA) technology. As time is divided into time slots and transactions within a time slot follow specific timing requirements which is shown in Figure 2.5 and is to be elaborated in Section 2.3.4, it is crucial that nodes in the network are kept in synchronization during its entire operation period.

Two mechanisms are applied in maintaining the network-wide time synchronization

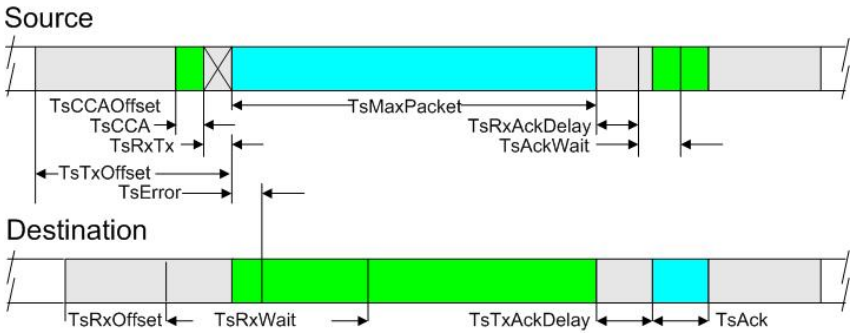

Figure 2.5: WirelessHART slot timing

in our network. When a node joins a WirelessHART network initially, it has no knowledge of the current time in the network. Fortunately, for each incoming DLPDU, a node records the time when the DLPDU's first bit arrives. Because of the strict time slot structure, a node can derive the start of the next time slot (T_s) from the DLPDU arrival time (T_a) according to the following formula:

$$T_s = T_a + 10ms - \text{TsTxOffset}$$

Synchronization happens not only in the join process, but also during a node's normal operations. A receiving node always compares the start time of the incoming DLPDU and the expected arrival time measured in its own clock. The difference is the drift between their clocks. The receiver includes the difference in the time adjustment field of the corresponding ACK packet. Each node is designated a time source node. Whenever a node receives an ACK from its time source, it adjusts its clock based on the time adjustment field. If the sender is the time source of the receiver, the receiver adjusts its clock directly from the time difference value.

2.3.3 Channel Blacklisting and Channel Hopping

Since 2.4 GHz band is becoming more and more crowded, to fine-tune the channel usage and improve the co-existence performance, WirelessHART applies the idea of *channel blacklisting*. Channels affected by consistent interferences could be put in the black list. In this way, the network administrator can disable the use of those channels in the black list.

The strict time synchronization in WirelessHART also makes the channel hopping technology [15] practical. It allows the communicating devices to rendezvous in hopping channels, thus providing frequency diversity and enhancing the communication reliability.

To support channel hopping, each device maintains an active channel table. Due to channel blacklisting, the table may have less than 16 entries. For a given slot and channel offset, the actual channel is determined from the following formula:

$$ActualChannel = (ChannelOffset + ASN) \% NumChannels$$

The actual channel number is used as an index into the active channel table to get the physical channel number. Since the ASN is increasing constantly, the same channel offset may be mapped to different physical channels in different slots. Thus we provide channel diversity and enhance the communication reliability.

2.3.4 Design Challenges and Solutions

Our overall architecture design of the data link layer is described in Figure 2.6 which includes five major modules. In the following of this section, we will describe the challenges in designing these modules and our solutions to tackle these problems.

Timer Module Design

Timer is the most fundamental module in our TDMA-based data link layer. It provides accurate timing to ensure the correct operating of the system. One significant challenge

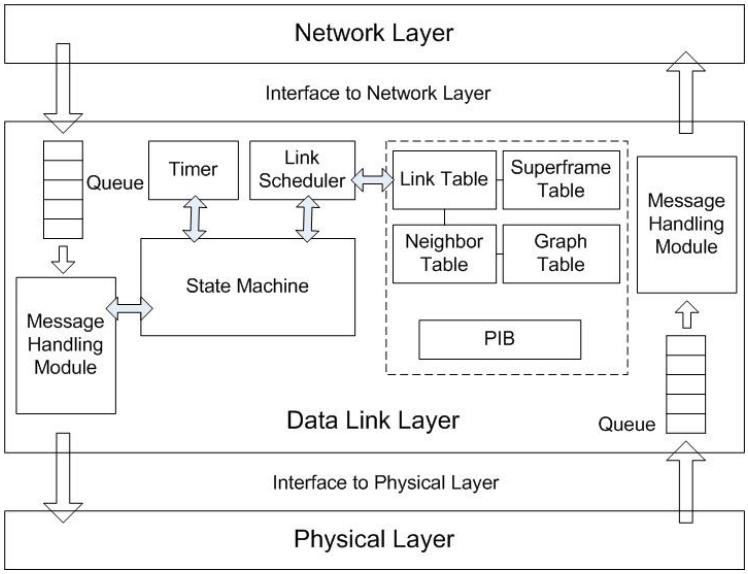

Figure 2.6: WirelessHART data link layer architecture

we met during the implementation is how to design the timer module and keep those 10ms time slots in synchronization. The specific timing requirement inside a WirelessHART time slot is depicted in Figure 2.5. When a node wants to send a frame, it first does a CCA check at *TsCCAOffset* time units after the start of the time slot. If the channel is clear, it starts to transmit the frame at *TsTxOffset*. After finishing sending the frame, it switches the transceiver from transmit (Tx) mode to receive (Rx) mode and waits for the acknowledgement. On the receiver side, the receiver waits *TsRxOffset* time units to listen for frames. After receiving the frame, it waits *TsTxAckDelay* to send out the acknowledgement.

WirelessHART has very stringent timing requirements on each network device. A 10ms time slot is further sliced into several time intervals, each of which ranges from 100μs to 4.5ms. For example, as shown in Figure 2.5, a receiver must start listening *TsRxOffset* time units after the beginning of a time slot. In addition, a receiver must acknowledge a packet within *TsTxAckDelay* time units after the arrival of the first bit of the packet. Some of the time intervals are very short. For instance, *TsCCA*, the CCA detection time, is defined to be 128μs.

Note that an ACK DLPDU is required to carry a 2-byte time adjustment field measured in microseconds. Thus, the timer used in WirelessHART MAC must be precise enough to count in microseconds.

During each time interval defined in a time slot, a node can either be idle or perform some tasks if necessary. Some of those tasks may be very time consuming. For example, when a node receives a DLPDU, it has to first verify the MIC (message integrity code) and then prepare the corresponding acknowledge. Ideally, those tasks should be finished within the designated time interval. However, in practice, the execution may take a longer time and by the time those tasks are completed, the predefined next time interval already starts. In this case, the subsequent tasks will be in serious trouble. Consequently, we cannot wait till the end of all tasks in current interval to set the next timer. Instead, we use the timer module to start/stop the tasks.

On our current hardware platform to be described in Section 4.1, we use a separate 16-bit TPM (Timer/Pulse Width Modulator Module) module to implement the timer. The TPM module's input clock is set to the bus clock (24MHz). By changing the internal prescaler of the TPM module, we can change the clock frequency of the timer. Currently, the prescaler is set to 24. As a result, each tick of the timer is $1\mu s$, which is precise enough to meet the data link layer timing requirement. The TPM module contains one free running counter and one comparison counter. Whenever the free running counter equals the comparison counter, a timer interrupt is triggered.

By adjusting the comparison counter and maintaining some internal data structures, the timer module can simulate several software timers. The caller of the timer module indicates what type the next time slot would be. Then the timer module generates a sequence of timeout events in the slot based on the given time slot type (transmit/receive/idle). As an example, if current slot is a *receive* slot, the timer would first generate an interrupt at the start of the slot. Then, *TsRxOffset* time units later, it would automatically generate another interrupt to the MAC, informing the MAC to put the transceiver in the listening

mode. Conceptually, an interrupt handler is composed of two parts: synchronous part and asynchronous part. The synchronous part resides in the interrupt handler, whereas the asynchronous part is included in the MAC state machine. Time critical and light-weight jobs are put in the synchronous part, and less time critical and computation intensive jobs are put in the asynchronous part. For the second interrupt in the example above, the interrupt handler only needs to set the mode of the transceiver and change some internal states, which can all be put in the synchronous part and leave the asynchronous part empty. However, an asynchronous part is needed when some time-consuming job is incurred, such as data encryption and decryption. In this case, at the very end of the interrupt handler, a specific event is sent to the MAC state machine to signal the execution of the asynchronous part. The interrupt handler finishes immediately after the signaling.

State Machine Design

The state machine in the data link layer consists of three primary components: the TDMA state machine, the XMIT and RECV engines. The TDMA state machine is responsible for executing the transaction in a slot and adjusting the timer clock. The XMIT and RECV engines deal with the hardware directly, which send and receive a packet over the transceiver, respectively. After the link scheduler decides the next slot to be serviced, it invokes the TDMA state machine, passing as parameters the transaction in the slot and the corresponding packet (if available). The TDMA machine handles the details of the transaction, such as the timing requirement in a time slot, calculating the message MIC (Message Integrity Code), sending/receving the DLPDU, and awaiting/sending the acknowledgement. Each run of the state machine contains three steps:

1. Call the link scheduler to determine the next slot to be serviced.

2. Receive the “time slot start” event from the timer and increment the ASN by 1.

3. When it is time to service the given time slot derived in step 1), execute the associated

transaction.

Most of the code in the state machine deals with executing a transaction. We define six states in the state machine:

- **Join:** In this state, the device is not authenticated by the Network Manager yet. After successfully joining the network, it enters the Idle state.
- **Idle:** When the device successfully joins the network or finishes transmiting/receiving a packet, it enters this state.
- **Talk:** When ready to transmit a packet, the state machine enters this state and calls the XMIT engine.
- **Wait ACK:** After a non-broadcast DLPDU is transmitted successfully, the state machine reaches this state.
- **Listen:** In this state, the state machine calls the RECV engine to wait for an incoming DLPDU.
- **Answer:** In this state, the state machine constructs and sends out an ACK DLPDU corresponding to the DLPDU received in the previous Listen state.

Based on these internal states and the incoming event type, the state machine knows what task to execute. For example, when it receives a "time slot start" event, it will first increase the ASN by 1. Then, if current slot is a *transmit* slot, it sets the transceiver to transmit mode and enters into the "Talk" state.

Communication Tables and Link Scheduler

Each network device maintains a collection of tables in the MAC layer. The superframe table and link table store communication configurations created by the Network Manager;

the neighbor table is a list of neighbor nodes that the device can reach directly and the graph table is used to collaborate with the network layer and store routing information.

In addition to these tables, a protocol information base (PIB) is created to keep track of the device's configuration parameters and current status. Various service primitives are provided by the interfaces to read and write these configuration information.

The functionality of the link scheduler is to determine the next slot to be serviced based on the communication schedule in the superframe table and link table. The scheduler is complicated by such factors as transaction priorities, the link changes, and the enabling and disabling of superframes. Every event that can affect link scheduling would cause the link schedule to be re-assessed.

The link scheduler first checks the DLPDUs in the outgoing queue and determines the first absolute slot number (ASN) that can be used to transmit a DLPDU. Next, it iterates through all receive links to determine the first ASN for listening. The smaller of the two ASNs selected above will be scheduled for servicing. Ties are resolved in favor of transmit slots.

Interface and Message Handling Module

As the data link layer sits between the physical layer and the network layer, the interface among them has to be defined clearly. Basically, the interface between the MAC and PHY layer describes the service primitives provided by the physical layer, and the interface between the MAC and network layer defines the service primitives provided to the network layer. This interface defines the MAC operations from a "black box" point of view.

The message handling module buffers the packets from the network layer and physical layer separately. As WirelessHART defines four priorities for DLPDU's, a DLPDU with higher priority must be inserted before a DLPDU with lower priority. In addition, this module supports message service primitives that locate a packet by the packet handle, *e.g.*, FLUSH.request and FLUSH.confirm.

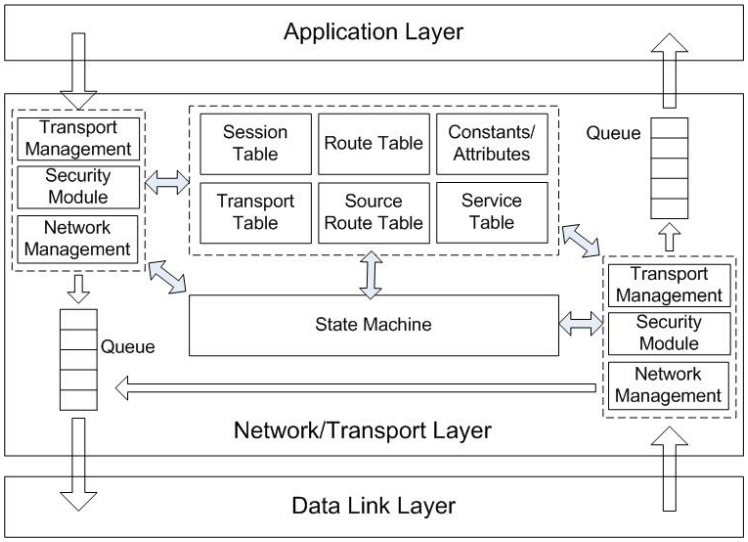

Figure 2.7: WirelessHART network and transport layer architecture

This module coordinates with the state machine and link scheduler to decide which packet for the specified link should be selected, enciphered and forwarded to its destination.

2.4 Network Layer and Transport Layer

WirelessHART supports a wide range of network topologies. The network layer and transport layer cooperate to provide secure and reliable end-to-end communication for network devices.

Figure 2.7 describes the detailed architecture of the network layer. For the purpose of security, WirelessHART is session oriented and a session enables confidential and secure communication between two end nodes. Each network protocol data unit (NPDU) would be enciphered before being sent out. At the destination, the NPDU is authenticated and deciphered. The details of the encryption, decryption and authentication mechanisms in WirelessHART is described in [81]. The transport layer of WirelessHART supports acknowledged operations which are used to construct a synchronous transport pipe across the network connecting devices. The transport pipe allows devices to send packets and confirm

their delivery in a fixed order.

2.4.1 Network Data Model Design

The network and transport layer maintain a set of tables, including the session table, transport table, route table and service table. These tables work together and collaborate with those in the MAC layer to achieve various functionalities. The interactions among these tables are summarized in Figure 2.8.

The session table is central in the design as all the end-to-end communications in WirelessHART are built upon secure sessions. A session establishes a secure data pipe between the device and one of its correspondent devices by enciphering and deciphering outgoing and incoming packets. Different types of sessions are assigned different security keys, such as session keys, join keys, and handheld keys.

The transport table is used to support end-to-end acknowledged transactions with automatic retries. In this way, WirelessHART provides reliable end-to-end communication between devices. The transport table uses a MASTER bit to identify whether the device is a MASTER or a SLAVE. Along with the corresponding sequence number, this table also buffers the payload of the last request (in MASTER mode) or response (in SLAVE mode). Thus it allows the device to retransmit the request or response when the retry timer expires.

For each destination, there can be more than one entries in the route table with different graph IDs. When generating a network layer packet, a node has to consult the route table, superframe table, and graph table together to determine the routing information to be used. For certain destination, there can also exist a source route, which is mainly used for network diagnostics.

Finally, the service table indicates the services associated with a certain route. All these services are allocated by the Network Manager and different services have different bandwidth requirements. It's the Network Manager's responsibility to carefully decide the communication schedule over the whole network to balance the traffic load.

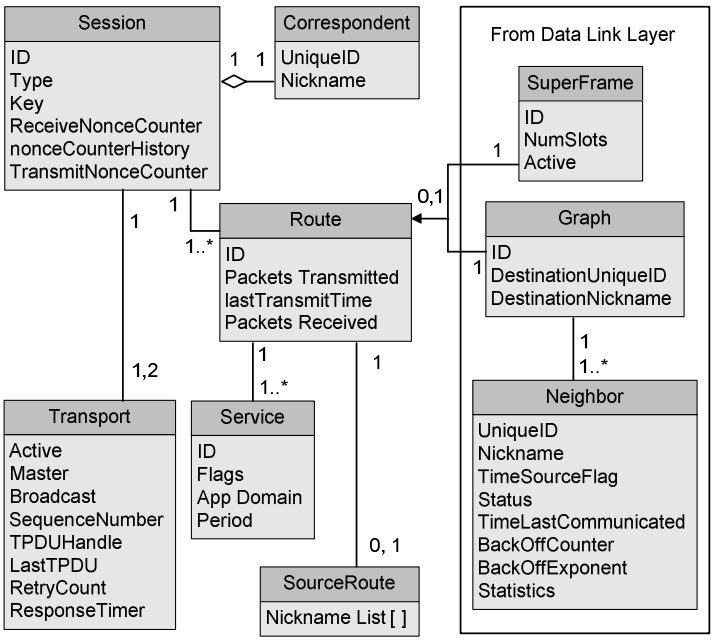

Figure 2.8: Network layer data model

With these well-organized communication tables and graph/source routing protocols, WirelessHART is able to establish secure and reliable sessions between the Gateway and any device. By this means, WirelessHART supports various network topologies and provides reliable end-to-end communications.

2.4.2 Routing Approaches

To support the mesh communication technology, each WirelessHART device is required to forward packets on behalf of other devices. There are three routing protocols supported in WirelessHART.

- **Graph Routing:** A graph is a collection of paths that connect network nodes. The paths in each graph are explicitly created by the Network Manager and downloaded to each individual network device. To send a packet, the source device writes a specific

graph ID (determined by the destination) in the network header. All network devices on the way to the destination must be pre-configured with graph information that specifies the neighbors to which the packets may be forwarded.

- **Source Routing:** Source routing is a supplement to the graph routing aiming at network diagnostics. To send a packet to its destination, the source device includes in the header an ordered list of devices through which the packet must travel. As the packet is routed, each routing device utilizes the next network device address in the list to determine the next hop until the destination device is reached. Otherwise the device at the point of failure must notify the Network Manager and discard the packet. It is the responsibility of the Network Manager to take corrective actions.

- **Superframe Routing:** Superframe routing is a special simplified version of Graph routing. When Superframe routing is performed, the Superframe ID is placed in the NPDU. Devices receiving a Graph Routed NPDU attempt to lookup the Graph ID, and failing that, the network layer looks for a superframe with the same value. If successful, the device may forward the packet to any neighbor with a link in that superframe.

In Chapter 3, we will describe how the Network Manager constructs and maintains the routing graphs to ensure the reliable routing between the peers and how to generate data link layer communication schedule in the mesh to achieve the end-to-end real-time performance for the data flows in WirelessHART networks.

2.5 Application Layer

The application layer is the topmost layer in the WirelessHART architecture. It defines various commands, data types and status. Since it is command-oriented, communications between peers are represented as command requests and responses. The application layer

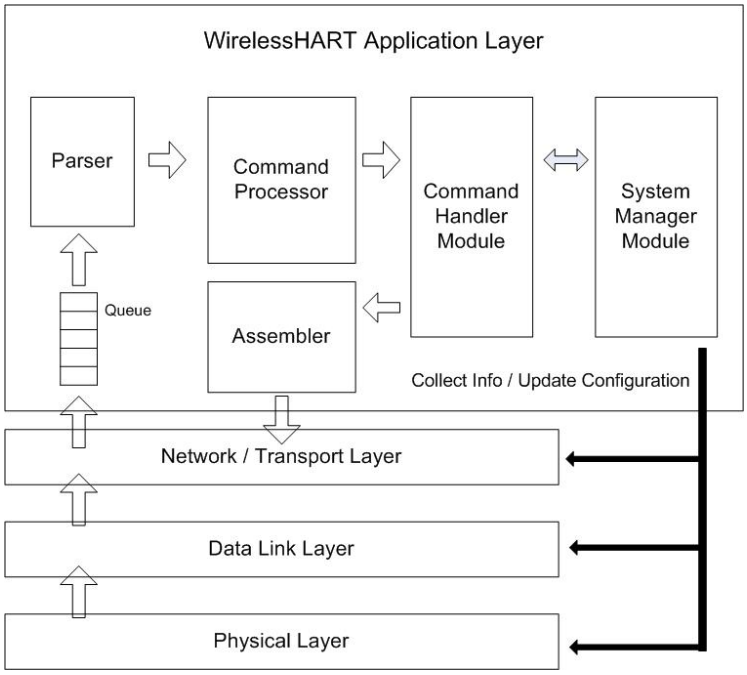

Figure 2.9: WirelessHART application layer architecture

is responsible for parsing the messages from its peer, extracting the command numbers, executing the specified commands, and generating responses.

Figure 2.9 describes our architecture of the application layer, whose core module is a command processor. When a field device receives an incoming command request, the application layer first parses the data message from the network layer into a command message (APDU) and relay it to the command processor. Based on the command number in the APDU, the command processor chooses the correct command handler to generate corresponding response. The command handler communicates with other layers through an intertwine service module to collect necessary information (for example, the number of links in the MAC layer) or update configurations (e.g., write a route to the network layer). Finally, the command handler assembles these information, generates the command response, and sends it back to the network layer.

The command number can be up to two bytes, which means as many as 65535

commands can be defined. The command number is followed by a one-byte *byte count* field and then the command payload. Currently, the WirelessHART specification has defined hundreds of commands. Although a WirelessHART device does not need to implement all of them, three classes of commands are mandatory for all devices.

- **Universal Commands** are used to provide the identification and software configuration information. For example, Command 0 is used to read the device's 5-byte unique ID.
- **Common Practice Commands** are mainly used to provide process data services like the burst data service.
- **Wireless Commands** are used to configure and maintain the WirelessHART network. For example, Command 965 is used to write superframes to the designated device.

Among these three classes, wireless commands are newly defined in the WirelessHART specification. These wireless commands wrap all services implemented by the network layer, the data link layer and the physical layer, and provide unified interfaces for the wireless network management.

2.6 Device Join Process

Before a new node can be integrated into a WirelessHART network, it must go through the join process. Through the join process, a node is provisioned with a short address (nickname), a network key for MAC layer authentication, a session key for network layer security, and most important, some links and routes to allow the bi-direction communications with the Network Manager.

To be able to start the join process, a new node should be first programmed with a network ID and a join key. The network ID determines which network the device will join,

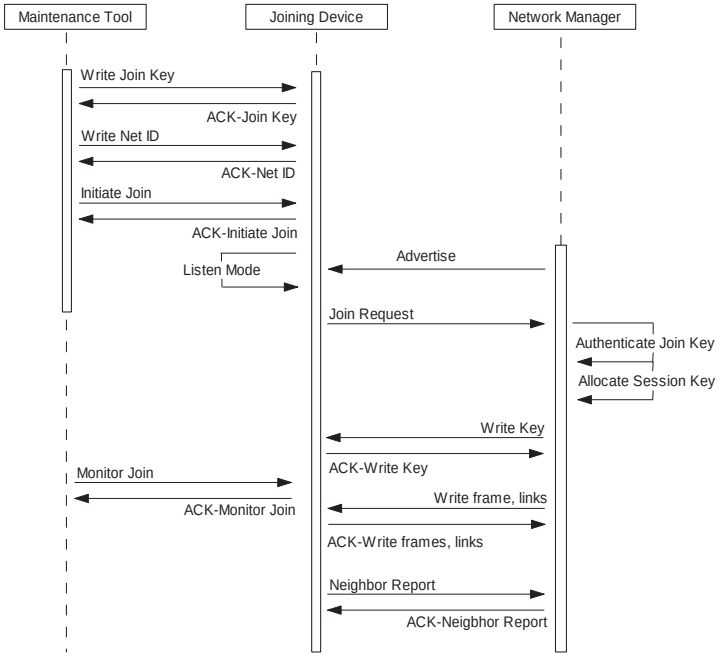

Figure 2.10: The join sequence in WirelessHART

while the join key is used to encipher the message payload before the device receives the session key. The Network Manager is pre-configured with the device's join key.

When the device is instructed to start the join process, the MAC layer is placed in a continuous listening mode. By capturing a network advertisement with the pre-programmed network ID, the node gets to know the start of a time slot and the network-wide absolute slot number. It can keep polishing its measurement of the start of a time slot with subsequently received advertisements. After the MAC layer fixes the timing, it starts to forward captured advertisements, together with the received signal strength, to the network layer. After gathering enough advertisements, the network layer decides which neighbor to send join requests to, usually based on the received signal strength and join priority associated with each advertisement.

Then the MAC layer assembles a join request and sends it through the join link as indicated in the advertisement from the selected neighbor. The selected neighbor then

forwards the join request to the Network Manager. The Network Manager checks the unique address and join key in the join request. If it deems the join request as valid, the Network Manager can grant the request by allocating a short address and sending the network key to the joining device. The new device shall use the network key thereafter. In addition, the Network Manager can write some superframes and associated links to the device, so that the new device can start to publish its sensing data periodically. This concludes the join process of a new node.

The critical step in the join process is time synchronization. As the advertisements are broadcasted using the channel hopping mechanism, the channels on which the new device listens should also change frequently. In our implementation, the device scans through all channels sequentially and stays on each channel for 400 ms. Another complication is the handling of channel blacklists. Special care must be taken if an advertisement contains a channel blacklist. In this case, the channels in the list should be skipped.

It is also important to switch from the join key to the network key and from the join links to normal links at correct time. It turns out there is a simple rule: always use the most updated resources. That is, a joining device keeps using the join key, until it receives a network key from the Network Manager; the device keeps sending/receiving packets on join links till it is configured with normal links.

2.7 Security Architecture

WirelessHART is a secure network system. Both the MAC layer and network layer provide security services. The MAC layer provides hop-to-hop data integrity by using MIC. Both the sender and receiver use CCM* mode together with AES-128 as the underlying block cypher to generate and compare the MIC.

The network layer employs various keys to provide confidentiality and data integrity for end-to-end connections. Four types of keys are defined in the security architecture:

- **Public Keys** which are used to generate MICs on the MAC layer by the joining devices.
- **Network Keys** which are shared by all network devices and used by existing devices in the network to generate MAC MIC's.
- **Join Keys** which are used during the joining process to authenticate the joining devices with the Network Manager. A join key is unique for each network device.
- **Session Keys** which are generated by the Network Manager. A session key is unique for each end-to-end connection between two network devices. It provides end-to-end confidentiality and data integrity.

Figure 2.11 describes the usage of these keys under two different scenarios: 1) a new network device wants to join the network, and 2) an existing network device is communicating with the Network Manager. In the first scenario, the joining device uses the public key to generate the MIC on MAC layer and uses the join key to generate the network layer MIC and encipher the join request. After the joining device is authenticated, the Network Manager creates a session key for the device and thus establishes a secure session between them. In the second scenario, on the MAC layer, the DLPDU is authenticated with the network key; on the network layer, the packet is authenticated and enciphered by the session key.

2.7.1 MAC Layer Security

In the MAC layer, WirelessHART provides data authentication service. The authentication service uses CCM* mode (Counter with CBC-MAC (corrected)) [22] with AES-128 [63] as the underlying block cypher. CCM* needs 4 byte-strings as parameters (a, m, N, K). As the DLPDU is not enciphered, the second parameter m is empty, while a includes the DLPDU header and payload.

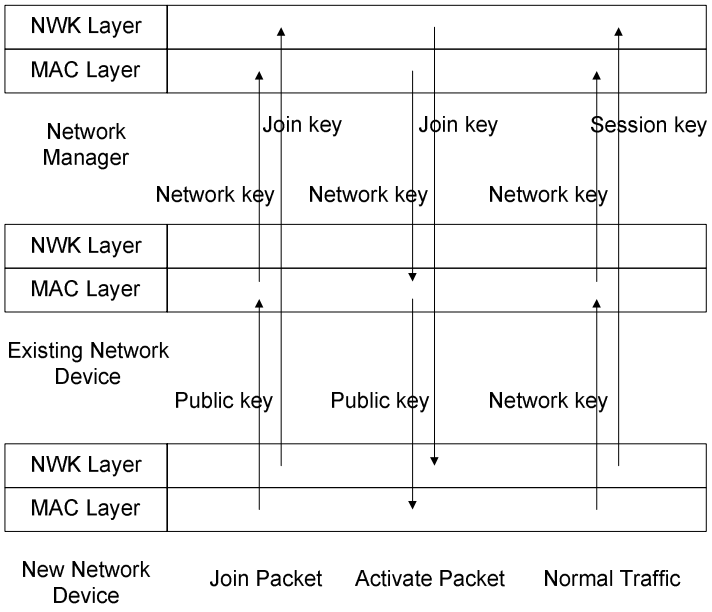

Figure 2.11: WirelessHART keying model

The key K is 16-byte long. The value of K depends on the current status of the node. If a node is joining a network or broadcasting a network advertisement, the well-known key 0x7777 772E 6861 7274 636F 6D6D 2E6F 7267 is used. In all other situations, the network key assigned by the Network Manager is used.

The nonce N is 13-byte long and is the concatenation of the absolute slot number (ASN) and the source address. The first 5 bytes are always the ASN. If the source address of the DLPDU is a long address (EUI-64 address), the source address is filled into the remaining 8 bytes of the nonce. Otherwise, the short source address (2 bytes) is put right next to the slot number, with the rest 6 bytes filled with 0.

The sender and receiver of the DLPDU both call the *CCM** function with the same input: the DLPDU header and payload. After receiving a DLPDU, the receiver compares the returned MIC with the MIC in the original message. If they match, the message is authenticated. Otherwise the message is invalid and discarded.

From the perspective of a receiver, it must run *CCM** on the received message and on the corresponding ACK message within *TsTxAckDelay (1ms)*. This is a very challenging task for low power processors.

We use hardware accelerator on our hardware platform to speed up the encipher/decipher process, and the encoding of 16 bytes of data would be finished in 13 system clocks. However, except for the *aes_encrypt()* function, *ccm_encrypt_message()* and *ccm_decrypt_message()* also call some other helper functions. The time taken by the help functions can not be improved by the hardware accelerator.

In order to further speed up the encipher/decipher process, we propose to execute *CCM** incrementally. Originally, *CCM** is not designed to support stream processing. However, as WirelessHART DLPDU is not enciphered, the message length is indicated in the DLPDU header. Thus, we can run *CCM** on an incoming message as soon as every 16 bytes are received. Given the relatively slow data transmission rate (250kbps), we may only need to process one block of data in the *TsTXAckDelay* period, regardless of the message length. In this way, we can meet the stringent timing requirements of WirelessHART.

2.7.2 Network Layer Encryption and Authentication

WirelessHART also provides built-in security support in the network layer. A keyed MIC is used to ensure that the NPDU arrives successfully and unmolested from the indicated source device. Similar to that in MAC layer, the MIC is generated and checked using *CCM** mode in conjunction with the AES-128 block cipher. For each session, four critical fields are kept up-to-date for NPDU encryption and authentication: *sessionKey* records the 128-bit write-only session key; *peerNonceCounter* maintains the largest nonce counter value received from the correspondent device; *myNonceCounter* is the nonce counter for packets sourced by the device, and *nonceCounterHistory* is an array of bits recording the nonce counters received. The most significant bit of *nonceCounterHistory* is always set and corresponds to the current *peerNonceCounter* value.

In the process of enciphering the NPDU, the NPDU payload to be enciphered is the byte-string *m*. The NPDU header is the byte-string *a*. In the byte-string *a*, the TTL, Counter, and MIC fields are set to zero and are replaced with their actual values before transmitting the packet. The corresponding 128-bit session key is the byte-string *K* and the 13-byte network layer nonce *N* is constructed differently depending on the transmission types.

If a packet is a join response, *N*[0] is set as 1 and we use the *peerNonceCounter* value (from the join request) as the nonce counter and load the joining device's EUI-64 address into the nonce. In other cases, we set *N*[0] as 0. *myNonceCounter* in the session would be pre-incremented by one and written to the nonce. Then the NPDU source address field (EUI-64 address or nickName) is loaded into the nonce.

To authenticate a received packet, at the destination device, the NPDU nonce is re-constructed and the packet is deciphered. If the NPDU is a join response, *N*[0] is set to 1, otherwise it is set to 0. The NPDU source address field is loaded into the nonce. If the message is a join request, the NPDU counter is four-bytes and would be copied to the nonce counter. On the other hand, if it is a join response, the NPDU counter would be compared to *myNonceCounter*. If they do not match, the packet is discarded. Otherwise, the *myNonceCounter* is copied to the nonce counter. Under all other cases, the NPDU counter is one-byte and the nonce counter is re-constructed. To do this, the most-significant three bytes of the *peerNonceCounter* are copied to *N*[1]-*N*[3]. If the NPDU Counter value is less than the quantity (1+LSB(peerNonceCounter) - sizeof(nonceCounterHistory)) then the 24-bit value in *N*[1]-*N*[3] is incremented. The NPDU Counter is copied to *N*[4].

The resulting nonce counter is compared to the nonceCounterHistory. If it corresponds to any of the bits set in the nonceCounterHistory, the packet must be discarded. Otherwise, the nonceCounterHistory will be updated accordingly. This finishes the packet authentication process.

2.8 Summary

In this chapter, we present the architecture and design details of a TDMA-based secure wireless communication protocol called WirelessHART. WirelessHART can serve as a good candidate for the network infrastructure of CPS subsystems to support secure, reliable and real-time services while taking energy saving into consideration as well. We share our first-hand experience in building a prototype communication stack. We describe several challenges we had to tackle during the implementation, such as the design of the timer, network-wide synchronization, communication security, and reliable mesh networking. For each challenge, we provide a detailed analysis and propose our solution.

Chapter 3

Supporting Reliable and Real-time Services in CPS subsystems

The real-time wireless communication protocol we presented in Chapter 2 provides a solid foundation for achieving reliable and real-time services in CPS subsystems. A typical CPS subsystem consists of sensors and actuators which are organized together and form a star, tree or mesh topology. To achieve reliable and real-time services in such networks, unlike the decentralized control adopted by wireless ad-hoc or peer-to-peer networks, we advocate explicit and centralized network management. This will push the complexity of ensuring reliable and real-time communication to a centralized Network Manager. Taking WirelessHART as an example, this chapter presents our solutions for tackling this challenge and shall explore efficient approaches for forming a wireless sensing and control network, managing reliable graph routing, allocating network resources and constructing data link layer communication schedules.

In a typical WirelessHART network, each sensor has a designated sample rate to publish its process data to the Gateway through multi-hop transmissions. In the other direction, the Network Manager sends the control data back to the actuators either on demand or periodically. To help relay different types of data, we define three types of communi-

cation graphs. The network shares one broadcast graph for propagating common control messages and one uplink graph for devices to publish process data. If needed, each device further has a unique downlink graph from the Network Manager for forwarding specific control messages to itself. Although several work in the literature has been devoted on the design of data link layer scheduling in WirelessHART networks [24, 73, 80, 100], how to satisfy the strict reliability requirements on the routing graphs and construct data link layer communication schedules on top of them is still a challenging problem and has not received sufficient attentions.

In this chapter, we first abstract the reliability requirements for packet routing defined in WirelessHART standard. We present efficient algorithms to construct these reliable graphs and describe the recovery mechanisms in the face of network dynamics. These algorithms are designed to maintain the maximum number of reliable nodes in the graphs while achieving good network latency. To improve the scalability of the downlink graphs in large-scale networks, we further propose an extension on the standard to replace the single downlink graph with a sequence of ordered local graphs. These local graphs work as reusable building blocks in constructing downlink graphs for different destinations thus greatly reduce the overall overhead in device configuration.

Based on these routing graphs, the data link layer communication schedule is further constructed. Our approach allows multiple devices to compete for the retry links to the same device, and split the traffic from one device among all its successors, thus reduces the bandwidth allocation on each of them. By designing the communication schedules on the successors so that their combination has the same communication pattern as the original device, the global communication schedule is then spliced into sub-schedules and distributed to the corresponding devices. These sub-schedules work together and guarantee that the periodic process/control data between devices and the Gateway can be forwarded through multi-hops in a timely manner.

We have conducted extensive experiments to evaluate the performance of the pro-

posed algorithms. We have also built a complete WirelessHART communication system, and integrated our network management solutions into the Network Manager. We are deploying the system in different testing environments and collecting long-term measurement data. The details of the system design, implementation and deployment will be presented in Chapter 4.

The remainder of this chapter is organized as follows. Section 3.1 reviews the previous works on reliable routing and real-time scheduling in WirelessHART networks. The details of reliable graph routing and communication schedule construction in WirelessHART are described in Section 3.2 and Section 3.3. Section 3.4 summarizes our experiment results. We conclude the chapter and discuss the future works in Section 3.5.

3.1 Related Work

In this section, we summarize previous works in the literature on achieving reliable routing in wireless networks, and describe recent works on link and channel scheduling in WirelessHART networks to achieve end-to-end real-time communication.

3.1.1 Reliable Routing in Wireless Networks

The reliable graph routing defined in WirelessHART standard is essentially a multipath routing approach which has been extensively studied in wireless networks, and recognized as an efficient approach for improving the routing reliability [62, 61, 26, 79, 56, 98]. In [26], node-disjoint and braided multipath schemes are proposed to provide energy efficiency and resilience against node failures. SMR [79] is a multipath version of DSR. It is designed to utilize multipath concurrently by splitting traffic onto two maximally disjoint routes. AOMDV [56] is a multipath, loop-free extension to AODV. It ensures that alternate paths at every node are disjoint, therefore achieves path disjointness without using source routing. AODVM [98] is another extension to AODV for finding multiple node-disjoint paths. It also proposes an infrastructure to include deployment of reliable nodes which can route on

multiple paths. This infrastructure can increase the number of node-disjoint paths between the source and the destination especially when they are far apart.

Most of these works focus on identifying multiple node or edge-disjoint paths to improve the routing reliability. However, to deal with much harsher and noisier industrial control environments, the WirelessHART standard defines more stringent requirements on routing reliability. Each intermediate node on the routing graph must have at least two neighbors to forward the traffic to the destination. For this reason, the works in the literature cannot be directly applied in WirelessHART networks, and new routing algorithms have to be designed.

3.1.2 Real-time Scheduling in WirelessHART Networks

Our research group has been active in helping to develop and validate the design concepts of WirelessHART in partnership with Emerson Process Management Corporation. Since the standard was ratified in 2007, several research works have been devoted to the link scheduling and channel assignment problems in WirelessHART networks to achieve end-to-end real-time communication [73, 80, 100]. In [80, 100], the convergecast scheduling problem is studied in linear network topologies. They formulate the problem as a mixed integer linear programming problem, and design algorithms based on different assumptions on devices' buffering capability. [73] considers a more general WirelessHART network model including arbitrary network topology and multi-path routing. It formulates the sensor-to-actuator real-time flow scheduling problem and proves that it is NP-hard. Based on a necessary condition for schedulability in WirelessHART networks, it proposes an optimal scheduling algorithm based on a branch-and-bound technique. A practical scheduling policy called Conflict-aware Least Laxity First algorithm is also proposed to achieve better scalability and handle network dynamics.

However, all these aforementioned works assume that the network layer routes have already been provided and focus on data link layer scheduling. The relationship between

the routes and the data link layer schedules are not thoroughly studied. In our work, we present a general framework for network management in WirelessHART networks. We shall study how to achieve reliable graph routing for different communication paradigms in WirelessHART network and further construct a data link layer communication schedule based on them. Our solution can be easily integrated into the Network Manager, so that the setup of an operational WirelessHART network is simple and prompt.

3.2 Reliable Graph Routing

In this section, we present the details how we define and achieve the reliable routing in CPS subsystems which take WirelessHART as the communication protocol. We first describe the primary routing approaches adopted in WirelessHART in Section 3.2.1. Section 3.2.3 abstracts the reliability requirements on packet routing, defines three types of reliable graphs for different communication purposes, and describes their properties. We discuss the difficulties in achieving completely reliable routing in Section 3.2.4. The algorithmic details to construct these routing graphs are presented in Section 3.2.5, Section 3.2.6, Section 3.2.7 and Section 3.2.8. We describe the recovery mechanisms in Section 3.2.9.

3.2.1 Source Routing and Graph Routing

Two primary routing approaches are defined in the WirelessHART standard: graph routing and source routing. When using graph routing, a network device sends packets with a graph id in the network layer header along a path to the destination. All devices on the way to the destination must be pre-configured with graph information that specifies the neighbors to which the packets may be forwarded.

With source routing, to send a packet to its destination, the source includes in the network layer header an ordered list of devices through which the packet must travel. As the packet is routed, each routing device utilizes the next network device address from the packet header to determine the next hop to use. Since packets may go to a destination

without explicit setup of intermediate devices, source routing requires knowledge of the complete network topology.

Since the source routing approach only establishes a fixed single path between the source and destination, any link or node failure will cut off their communication. For this reason, source routing is mostly used for diagnostics purposes. In this chapter, we will focus on the graph routing approach and investigate how to achieve reliable routing in the network. Based on different communication purposes, there are three types of routing graphs defined in a WirelessHART network, and Figure 3.1 illustrates an example.

Uplink graph: It is a graph connecting all devices upward to the Gateway. It is used to propagate devices' process data periodically to the Gateway. Different devices may have different sample rates.

Broadcast graph: It is a graph connecting the Gateway downward to all devices. It is used to broadcast common configuration and control messages to the entire network.

Downlink graph: It is one per device. It is the graph from the Gateway to each individual device. The unicast messages from the Gateway and the Network Manager to each device traverse through this graph.

Based on these graphs, the Network Manager can further generate the corresponding sub-routes for each device. Only after the routes are constructed and downloaded to every device, can the network communication schedule be generated, which we shall elaborate in Section 3.3. When devices initially join into the network, they carry with them a list of neighbor entries including the received signal strength information. The Network Manager uses this information and the periodic health reports from the devices to construct and maintain the global network topology.

3.2.2 Notations

This section summarizes the notations to be used throughout this chapter. Given the original network topology $G(V, E)$, we use g to denote the Gateway, V_{AP} to denote the set of Access

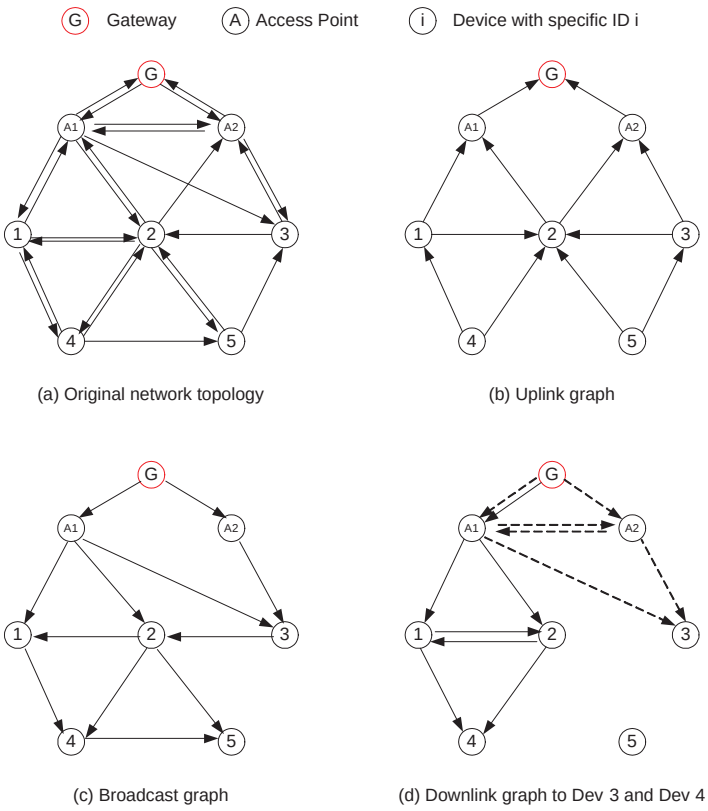

Figure 3.1: Three types of routing graphs

Points and V_D to denote the set of devices. We have $\{g\} \cup V_{AP} \cup V_D = V$. For each node $i \in V_D \cup V_{AP}$, we use e_i^+ and e_i^- to denote its set of outgoing edges and incoming edges. We use δ_i^+ and δ_i^- to denote its outgoing and incoming degree. $G_B(V_B, E_B)$ and $G_U(V_U, E_U)$ are used to represent G's reliable broadcast graph and uplink graph. The reliable downlink graph for node $v \in V$ is denoted by $G_v(V_v, E_v)$.

3.2.3 Reliability Requirements and Reliable Graphs

Compared with wireless community networks, WirelessHART network have a much higher demand on the routing reliability to tolerate node and link failures. In this section, we abstract the reliability requirements defined in WirelessHART standard using the concept of (k, m)-*reliability* in packet routing. Notice that here we assume that the Gateway and Access Points are all connected through wire and reliable, so in the following of the chapter, the reliability requirements only apply to wireless devices.

Definition 3.2.1: Given a directed graph $G(V, E)$, a node $v \in V$ satisfies the (k, m)-*reliability* if and only if $\delta_v^- \geq k$ and $\delta_v^+ \geq m$. There is no constraint on δ_v^- or δ_v^+ if $k = 0$ or $m = 0$.

Based on this definition, we now give the definitions of the aforementioned three reliable routing graphs and present their important properties.

Definition 3.2.2: Given a directed graph $G(V, E)$, a directed acyclic graph $G_B(V_B, E_B)$ ($V_B = V$ and $E_B \subseteq E$), is a reliable broadcast graph if the $(2, 0)$-reliability holds on every node in $V - \{g\} - V_{AP}$.

G_B requires that each device has at least two parents from which it can receive the broadcast messages. This significantly increases the chance for the broadcast messages to be propagated to the entire network. G_B has the following property.

Property 3.2.1: Each device in G_B has at least two paths from g.

Proof: According to the definition of G_B, $\forall v, v \in V - \{g\} - V_{AP}$, it has two different parent nodes. There are two cases on v's parent node u. In the first case, u is an Access Point. It is obvious that there exists one path $g \rightarrow u \rightarrow v$. In the second case, u is a device. We perform the same analysis on u and find its parents. As G_B is acyclic, this process can be repeated and terminates when it reaches an Access Point. Thus there exists a path $g \rightarrow \cdots \rightarrow u \rightarrow v$. Because v has two different parent nodes, there are at least two paths from g to v in G_B. □

Different from the broadcast graph, the uplink graph is used by the devices to forward their process data to the Gateway with a required sample rate. It is considered reliable if and only if for each device in the network except the Access Points, it has two children to forward its packet to the Gateway. In cases where the communication between the device and one of its children is broken, the process data can still be delivered to the Gateway through the alternative child.

Definition 3.2.3: Given a directed graph $G(V, E)$, a directed acyclic graph $G_U(V_U, E_U)$ ($V_U = V$ and $E_U \subseteq E$), is a reliable uplink graph if the $(0, 2)$-reliability holds on every node in $V - \{g\} - V_{AP}$.

Property 3.2.2: Each device in G_U has at least two paths to g.

Proof: The proof is similar to the proof for Property 3.2.1.□

Property 3.2.3: G_B and G_U both have no less than 2 Access Points.

Proof: Assume that there is only one Access Point p in G_B. and v is a node with an incoming edge from p in G_B. As p is the only Access Point, node u, the other parent node of v is a device. We repeat this analysis on u and it is obvious that at least one of u's parents is still a device. This process will be repeated until a cycle is formed because the number of devices in the network is finite. This is a contradiction with the definition of G_B. So G_B has no less than 2 Access Points. The proof for G_U is similar. □

The broadcast graph and uplink graph are global graphs shared by the entire network. However, to support the transmission of configuration and control messages to a

specific device v, a unique downlink graph $G_v(V_v, E_v)$ from the Gateway and Network Manager to v is also required. G_v is defined to be reliable only if $(0, 2)$-reliability holds on each intermediate node.

Property 3.2.4: $G_v(V_v, E_v)$ contains at least one directed cycle.

Proof: Assume there is no cycle in G_v. Consider the node u which has a direct edge to v in G_v. According to the definition of G_v, intermediate node u has another non-v child w. Repeat this analysis on w, and w also has a non-v child. This process can be repeated and finally form a cycle. □

Property 3.2.4 states the existence of directed cycles in G_v. To guarantee the prompt delivery of the downlink messages, we must avoid arbitrary cycles in G_v which will generate infinite loops in packet forwarding. Thus in its definition, we restrict that there is only one cycle of length 2 in G_v and require that every node on the cycle must be the destination's parent. Once the packet reaches such nodes, it will be directly forwarded to the destination which is required by the standard. This will avoid any cyclic transmission and unnecessary delay.

Definition 3.2.4: Given a directed graph $G(V, E)$, a directed graph $G_v(V_v, E_v)$ ($V_v \subseteq V$ and $E_v \subseteq E$), is a reliable downlink graph from g to v if 1) v is the only sink and g is the only source in G_v; 2) $(0, 2)$-reliability holds on each intermediate node in G_v; and 3) there is only one cycle of length 2 in G_v, and each node on the cycle has a direct edge to v.

3.2.4 Difficulties in Achieving Completely Reliable Graphs

The major barrier to construct reliable routing graphs is the underlying network connectivity. Better network connectivity will obviously lead to a higher chance for constructing completely reliable graphs. In this section, we evaluate the relationship between the network connectivity and the success ratio to construct these reliable graphs. In our experiments, we vary the network connectivity by changing the edge success probability p and Figure 3.2

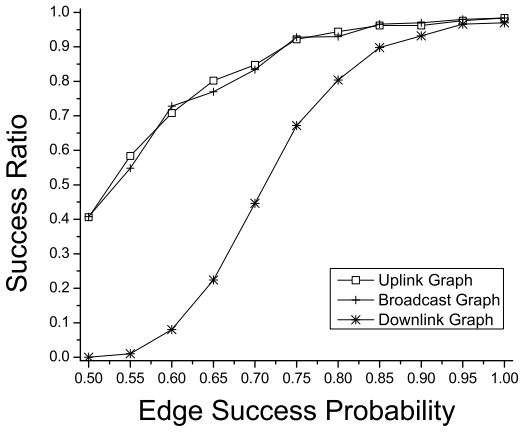

Figure 3.2: Success ratio vs. Edge success probability

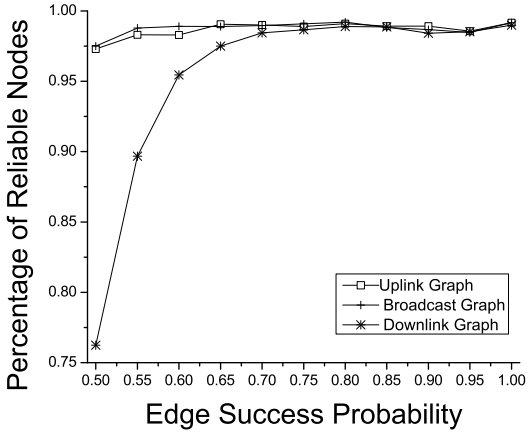

Figure 3.3: Percentage of reliable nodes

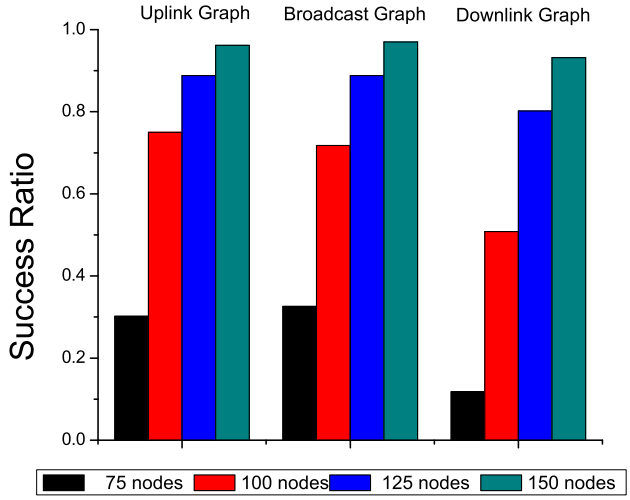

Figure 3.4: Success ratio vs. Network density

summarizes our results. We observe that with 150 devices in the experiments, the success ratio drops quickly along with the decrease of p. When p is 0.8, the success ratio is around 80% for downlink graphs and above 95% for both G_B and G_U. However, when p drops to 0.5, we can barely construct reliable downlink graphs and the success ratios for G_B and G_U are only around 40%.

Under the same experiment settings, Figure 3.3 shows the percentage of reliable nodes in the incomplete reliable graphs. We observe that the percentage of reliable nodes in incomplete G_U and G_B are always above 95% and this percentage for downlink graphs is also larger than 75% even when the edge success probability drops to 0.5. Figure 3.4 further evaluates the impact of the network density on the success ratio. We vary the size of the network from 75 to 150 and fix the edge success probability at 0.8. As expected, The results show that the network density has a great impact on network connectivity, and lower network density will lead to poor success ratio.

Based on these results, we conclude that the success ratio for constructing reliable routing graphs is closely related to the underlying network connectivity. In many scenarios, it is impossible to achieve the completely reliable graphs. For this reason, we shall allow violations of the reliability requirements in the routing graphs and instead focus on designing algorithms to construct graphs with the maximum number of reliable nodes. In the following of this chapter, we will still use G_B, G_U and U_v to represent the broadcast graph, uplink graph and downlink graph for node v even though they may not be completely reliable.

3.2.5 Constructing Reliable Broadcast Graph

In a broadcast graph, we say that a node i is reliable if and only if $\delta_i^- \geq 2$. Let $S_B = \{i \,|\, \delta_i^- \geq 2, i \in V\}$, and we want to maximize $|S_B|$ when we construct the reliable broadcast graph G_B. Furthermore, to reduce the energy consumption in propagating broadcast messages to the entire network and improve network latency, we also hope to minimize the average number of hops from the Gateway to each node. For node i, we denote its average number of hops

Alg 1 Constructing Reliable Broadcast Graph $G_B(V_B, E_B)$

// $G(V, E)$ is the original graph
Initially $V_B = g \cup V_{AP}$ and E_B contains all links from g to V_{AP}. For each AP i in V_{AP}, $\bar{h}_i = 1$

while $V_B \neq V$ **do**
  Find $S' \subseteq V - V_B$: $\forall v \in S'$, v has at least two edges from V_B
  if $S' \neq \emptyset$ **then**
    for all node $v \in S'$ **do**
      Sort its edges $e_{u,v}$ from V_B according to $\bar{h}_u$
      Choose the first two edges $e_{u_1,v}$ and $e_{u_2,v}$
      $\bar{h}_v = \frac{\bar{h}_{u_1} + \bar{h}_{u_2}}{2} + 1$
    end for
    Choose the node v with min $\bar{h}_v$
    Add v to V_B and add $e_{u_1,v}$ and $e_{u_2,v}$ to E_B
  else
    Find $S'' \subseteq V - V_B$: $\forall v \in S''$, v has one edge $e_{u,v}$ from V_B
    if $S'' \neq \emptyset$ **then**
      for all node $v \in S''$ **do**
        $\bar{h}_v = \bar{h}_u + 1$
        Calculate n_v, the # of its outgoing edges to $V - V_B$
      end for
      Choose the node v with maximum n_v, break tie using $\bar{h}_v$
    else
      return FAIL;
    end if
  end if
end while
return SUCCESS;

from the Gateway by $\bar{h}_i$ and use P_i to represent its parents in G_B. We have:

$$\bar{h}_i = \frac{\sum_{k \in P_i} \bar{h}_k}{|P_i|} + 1 \tag{3.1}$$

We present a greedy algorithm (Alg. 1) to achieve these two goals in constructing G_B. In our approach, we maintain a set V_B to record the explored nodes and V_B is initialized as $\{g\} \cup V_{AP}$. The explored edges are maintained in E_B which is initialized to include the edges from g to each Access Point. In the algorithm, we incrementally select one node v from $V - V_B$. In each loop, we first find S', the set of reliable nodes in $V - V_B$ (Line 5). For each node v in S', we sort its incoming edges from V_B according to their averaged number of hops from the Gateway in ascending order. We choose the first two edges and calculate $\bar{h}_v$ according to Eq. 3.1. We choose the node in S' with the minimum $\bar{h}_v$ and add it to V_B. If there is no reliable node available in $V - V_B$, we will instead find S'', the set of nodes with exact one edge from V_B (Line 15). We choose the node in S'' with the maximum number of outgoing edges to $V - V_B$ to maximize the chance to find reliable nodes in the next round. This process continues until all nodes in V are explored. Otherwise an error will be reported (Line 24). This will trigger the Network Manager to execute appropriate recovery actions. The *worst-case* complexity of the algorithm is

$$\sum_{k=|V_{AP}|}^{|V|} (|E| + (|V| - k) \cdot \lg(|V| - k)) = O(|V|^3)$$

3.2.6 Constructing Reliable Uplink Graph

The construction of a reliable uplink graph $G_U(V_U, E_U)$ is similar to that of $G_B(V_B, E_B)$. Essentially, we only need to reverse all edges in the original graph $G(V, E)$, construct G_B and then reverse all its edges back. We define $G^R(V, E^R)$ to be the reversed graph of $G(V, E)$, and the greedy algorithm to construct $G_U(V_U, E_U)$ is summarized in Alg. 2 and its *worst-case* complexity is $O(|V|^3)$.

Alg 2 Constructing Reliable Uplink Graph $G_U(V_U, E_U)$

```
1: // G(V, E) is the original graph, G^R(V, E^R) is the reversed graph
2: Construct G^R(V, E^R)
3: Construct G_B(V_B, E_B) from G^R(V, E^R) by applying Alg. 1
4:
5: if V_B = V then
6:     // Construct G_U by reversing all edges in G_B
7:     G_U(V_U, E_U) = G_B^R(V_B, E_B^R)
8: else
9:     // the network topology is disconnected
10:    return FAIL;
11: end if
12: return SUCCESS;
```

3.2.7 Constructing Reliable Downlink Graph

The construction of the reliable downlink graph $G_v(V_v, E_v)$ for a given node v in $G(V, E)$ only involves part of the nodes in $G(V, E)$ and it is more complicated because of the existence of cycles as shown in Property 3.2.4. Furthermore, according to Definition 3.2.4, we want to have exactly one cycle in G_v of length 2 and restrict it to be between the two parents of v. Our optimization goals in constructing G_v are similar to that of G_B and G_U. We hope to maximize the number of nodes in the network to have reliable downlink graphs and for each downlink graph, we want to minimize its average number of hops from the Gateway.

Alg. 3 summarizes the framework of our approach. In the algorithm, we construct the reliable downlink graph for each node in the network. For the Access Point, its downlink graph consists of the Gateway g, itself and the edge from g to itself. We maintain S, a set of nodes whose reliable downlink graphs have already been constructed (Line 1). We incrementally find an eligible node v in $V - S$ to construct G_v where three constraints in Table 3.1 are applied and v has the minimum $\bar{h}_v$ as calculated in Line 17. Constraint C1 and C2 are to satisfy the reliability requirements in G_v and Constraint C3 is to make sure that we can remove the internal cycles in the constructed G_v. If such an eligible node cannot be found, we will instead choose the node that has two parents from S with the minimum

C1: v has at least two parents u_1 and u_2 in S
C2: u_1 and u_2 form a directed cycle
C3: u_2 (u_1) has at least one parent from the cycle in G_{u_1} (G_{u_2})

Table 3.1: Three constraints in constructing reliable downlink graphs

average latency to the Gateway (Line 20). If every node in $V - S$ only has one parent from S, we choose the one with the minimum average latency (Line 27 - 37).

Alg. 4 describes how we construct G_v based on its parents (u_1 and u_2)' reliable downlink graphs G_{u_1} and G_{u_2}. We first merge G_{u_1}, G_{u_2}, v and edges among u_1, u_2 and v together (Line 4). We maintain S, the set of explored nodes in G_v and initialize it as $\{g, v, u_1, u_2\}$. We construct G_v in a bottom-up manner by incrementally selecting a node $i \in V_v - S$ which has two outgoing edges to S in G and has the minimum $\bar{h}_i$ (Line 6-30). This process continues until either all nodes in V_v are explored or V_{AP} has two outgoing edges to S (Line 7 - 10). Finally, we remove all nodes in $V_v - S$ and their corresponding edges from G_v (Line 32 - 34). If there is no node available to have two outgoing edges to S in G, we choose the node with the minimum $\bar{h}_i$ (Line 20 - 29).

3.2.8 Constructing Scalable Reliable Downlink Graph

The algorithms proposed in Section 3.2.7 strictly comply to the WirelessHART standard and construct one downlink graph for each individual node. However, this approach is not scalable. When a device is multi-hop away from the Gateway, its downlink graph has to traverse multiple intermediate devices but cannot reuse their downlink graph information. This will introduce unnecessarily high configuration overhead in the network. To achieve reliable downlink graph routing in large-scale wireless networks, in this section we propose to extend the current downlink route from a single graph to a sequence of ordered local graphs, and we call this approach Sequential Reliable Downlink Routing (SRDR). Instead of constructing a completely new graph from Gateway to device v, SRDR lets each node only keep a small local graph to maintain the reliable routing from its parents. The reliable

Alg 3 Constructing Reliable Downlink Graphs in $G(V, E)$

Let S be the set of nodes with downlink graphs constructed
Initially $S = g \cup V_{AP}$ and $G_g = (\{g\}, \emptyset)$
Initially for each AP i in S, set $G_i = (\{g \cup i\}, \{e_{g,i}\})$

while $S \neq V$ **do**
  Find $S' \subseteq V - S$: $\forall v \in S'$, v has at least two edges from S
  // S_r is the reliable node set in S', initially $S_r = \emptyset$
  if $S' \neq \emptyset$ **then**
    for all node $v \in S'$ **do**
      for all edge pair $(e_{u_1,v}, e_{u_2,v})$ from S **do**
        if C 1 $\wedge$ C 2 $\wedge$ C 3 **then**
          $S_r = S_r \cup \{v\}$
        end if
        $\bar{h}_{u_1,u_2} = (\bar{h}_{u_1} + \bar{h}_{u_2})/2$
      end for
      Choose the edge pair $(e_{u_1,v}, e_{u_2,v})$ with min $\bar{h}_{u_1,u_2}$
      $\bar{h}_v = \bar{h}_{u_1,u_2} + 1$
    end for
    if $S_r \neq \emptyset$ **then**
      Add node v in S_r with min $\bar{h}_v$ to S
    else
      Add node v in S' with min $\bar{h}_v$ to S
    end if
    // construct G_v: $\bar{h}_{u_1,u_2}$ is the min among all edge pairs to v
    ConstructDG(G, G_{u_1}, G_{u_2}, v);
  else
    Find $S'' \subseteq V - S$: $\forall v \in S''$, v has one edge $e_{u,v}$ from S
    if $S'' \neq \emptyset$ **then**
      for all node $v \in S''$ **do**
        $\bar{h}_v = \bar{h}_u + 1$
        Calculate n_v, the # of v's outgoing edges to $V - S$
      end for
      Add v to S with maximum n_v, break tie using $\bar{h}_v$
      ConstructDG(G, G_{u_1}, null, v);
    else
      return FAIL;
    end if
  end if
end while
return SUCCESS;

Alg 4 ConstructDG ($G(V, E)$, $G_{u_1}(V_{u_1}, E_{u_1})$, $G_{u_2}(V_{u_2}, E_{u_2})$, v)

Let S contain explored nodes in $G_v(V_v, E_v)$: $S = \{g, v, u_1, u_2\}$

// Construct G_v: Merging G_{u_1}, G_{u_2}, v, and edges between u_1, u_2 and edges from u_1 and u_2 to v

$G_v(V_v, E_v) = G_v(V_{u_1} \cup V_{u_2} \cup \{v\}, E_{u_1} \cup E_{u_2} \cup \{e_{u_1,v}, e_{u_2,v}, e_{u_1,u_2}, e_{u_2,u_1}\})$

while $S \neq V_v$ **do**
  if V_{AP} has two outgoing edges to S in G **then**
    $S = S \cup V_{AP}$
    break;
  end if
  for all node $i \in V_v - S$ **do**
    Sort i's outgoing edges to S in descending order of $\bar{h}_i$
  end for

  Find $S' \subseteq V_v - S$: $\forall v \in S'$, v has at least two edges to S
  if $S' \neq \emptyset$ **then**
    Add node i with min $\bar{h}_i$ to S
    Add first two edges from i to S to G_v if they don't exist
    Remove all other edges from E_v
  else
    Find $S'' \subseteq V_v - S$: $\forall v \in S''$, v has one edge to S
    if $S'' \neq \emptyset$ **then**
      Add i with min $\bar{h}_i$ to S
      Add the edge from i to S to G_v if it doesn't exist
    else
      return FAIL;
    end if
  end if
end while

for all node $i \in V_v - S$ **do**
  Remove i from V_v and corresponding edges from E_v
end for
return SUCCESS;

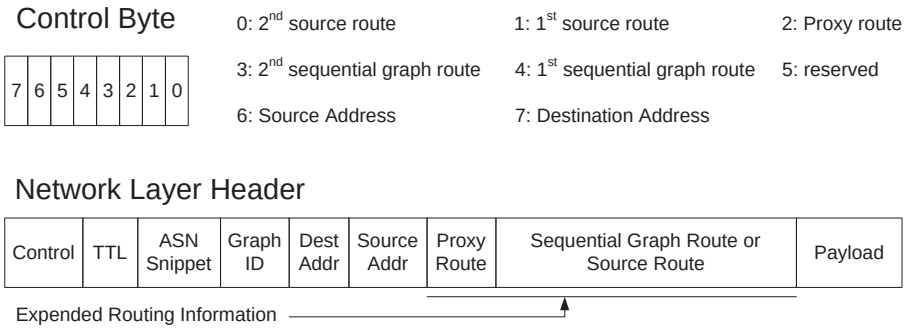

Figure 3.5: The extension of the network layer header in WirelessHART to support sequential reliable downlink routing

downlink graph to a given node can be constructed by assembling the intermediate nodes' local graphs together based on a given order. These local graphs can be taken as building blocks in constructing downlink graphs for different destinations, thus existing device configurations can be reused. This will significantly reduce the overall configuration overhead and improve the downlink routing scalability.

Extension: To support sequential reliable downlink routing, we need two extensions in the current WirelessHART standard. First, as depicted in Figure 3.5, we use the reserved bits (Bits 4-3) of the control byte in the network layer header to indicate, when set, the presence of the sequential downlink routing fields, and we use the source routing option field to store the ordered graph list; Second, the routing module is enhanced to support SRDR. When the packet arrives at the intermediate node, the routing module will retrieve the earliest graph ID in the graph list and verify if the current node is the sink of this specific graph. If it is, we remove this graph ID from the graph list and route this packet on the next earliest graph. This process continues until we reach the final destination or the routing fails. In the latter case, we will remove this graph ID and try the next earliest graph ID if it has the corresponding edges. Otherwise, alarm messages will be sent to the Network Manager and appropriate actions shall be taken.

Alg. 5 summarizes the framework of SRDR. In the algorithm, given the original

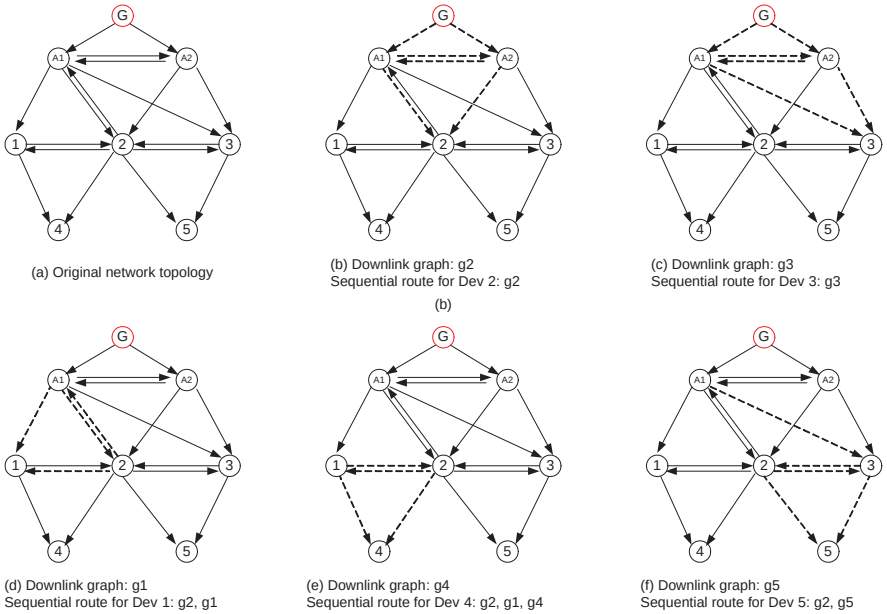

Figure 3.6: Examples of the sequential reliable downlink routes

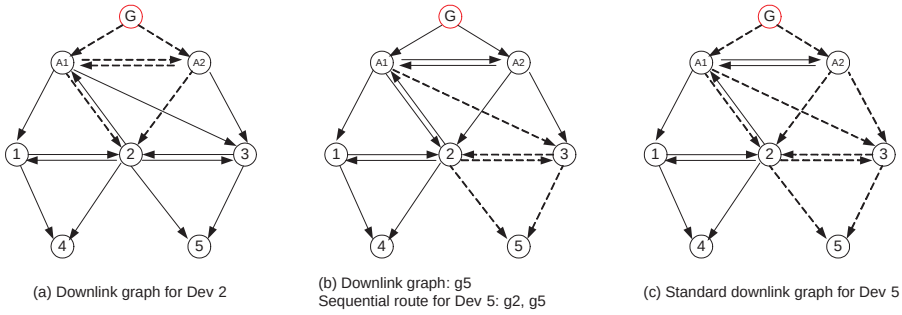

Figure 3.7: Standard approach vs. Sequential reliable downlink routing (SRDR)

C1: v has at least two parents u_1, u_2, and they form a cycle.
C2: u_1 is u_2's parent in u_2's local downlink graph.
C3: u_2 (u_1) has at least one parent from the cycle in G_{u_1} (G_{u_2})

Table 3.2: Three constraints in constructing scalable reliable downlink graphs

graph G, we construct the reliable downlink route (an ordered graph list) for each node in the network. For the Access Point, its downlink route contains only one local graph which consists of the Gateway g, itself and the edge between them. We maintain S, a set of nodes whose downlink routes have already been constructed (Line 1). We incrementally find an eligible node v in $V - S$ to construct its downlink route R_v where three constraints in Table 3.2 are applied and v has the minimum $\bar{h}_v$ as calculated in Lines 14-26. Constraint C1 is to find v's local downlink graph $g_v = (\{u_1 \cup u_2 \cup v\}, \{e_{u_1,u_2}, e_{u_2,u_1}, e_{u_1,v}, e_{u_2,v}\})$; If constraint C2 is satisfied, v's downlink route R_v can be simply derived as $R_v = R_{u_2} \rightarrow g_v$; Constraint C3 presents another way to construct the reliable downlink route for v if u_1 and u_2 are independent. If an extra edge e can be found from the cycle in G_{u_1} to u_2 or from the cycle in G_{u_2} to u_1, it will be added into g_v, and R_v can be derived as $R_{u_1} \rightarrow g_v$ or $R_{u_2} \rightarrow g_v$. If such an eligible node cannot be found, we will instead choose the node that has two parents from S with the minimum average latency to the Gateway (Line 18). If every node in $V - S$ has only one parent from S, the one with minimum average latency will be chosen (Line 28 - 40). Alg. 6 gives the details how we construct R_v.

Example 3.2.1: Figure 3.6 illustrates an example for constructing the reliable downlink routes for devices in a WirelessHART network. Figure 3.6(a) gives the original topology of the network. We first include node 2 and node 3 into the explored node set S. The dotted lines in Figure 3.6(b) and Figure 3.6(c) show their local downlink graphs. When adding node 1 into S, as $A1$ and node 2 are already in S and they satisfy the constraints C1 $\wedge$ C2, R_1 is derived as $g_2 \rightarrow g_1$. We have the similar operations when adding node 4 into S and $R_4 = g_2 \rightarrow g_1 \rightarrow g_4$. However, when we add node 5 into S, node 2 and node 3 are

independent. As we have a link between $A1$ and node 3, constraints C1 $\wedge$ C3 are satisfied. The dotted links in Figure 3.6(f) shows g_5, and the downlink route of node 5, R_5 is $g_2 \rightarrow g_5$.

The next example compares the standard approach in WirelessHART with sequential reliable downlink routing (SRDR).

Example 3.2.2: Figure 3.7 compares SRDR with the standard approach in WirelessHART. The downlink graphs for node 2 under both approaches are the same (Figure 3.7(a)). The downlink route for node 5 in our approach is $R_5 = g_2 \rightarrow g_5$, and g_5 is shown in Figure 3.7(b). In SRDR, the downlink routing from the Gateway to node 5 can leverage the local routing graph in intermediate node (node 2) while only a local graph in node 5 is needed. However, the standard approach has to construct a completely new graph from the Gateway to node 5 which is shown in Figure 3.7(c). Comparing Figure 3.7(b) and Figure 3.7(c), the standard approach requires 3 extra links to achieve the reliable downlink routing. This overhead will increase dramatically when the destination is far away from the Gateway.

Optimization: In the basic SRDR, the routing is performed strictly according to the sequence in the ordered graph list. However, as each node can keep graph information to multiple destinations, we can take advantage of the "shortcut" to further improve the network latency. We call this approach SRDR-OPT. When a packet arrives at an intermediate node i, instead of using the earliest graph ID, SRDR-OPT searches the ordered graph list *backward* and finds the first graph ID that is stored in its routing table. The packet then will take the "shortcut" and be forwarded on this graph. If this forwarding is successful, at the destination of this selected graph, all the preceding graph IDs in the ordered graph list including the current ID will be removed. Otherwise, node i will choose the next available graph ID *backward* in the ordered graph list and repeat this process. The following example shows the advantage of SRDR-OPT.

Example 3.2.3: In Figure 3.8, we are routing packets from node s to node 4 and R_4 is

Alg 5 Constructing Sequential Reliable Downlink Routes

Let S be the set of nodes with downlink route constructed
Initially $S = g \cup V_{AP}$
For each AP i in S, set $G_i = (\{g \cup i\}, \{e_{g,i}\})$, $R_i = G_i$ and $\bar{h}_i = 1$

while $S \neq V$ **do**
  Find $S' \subseteq V - S$: $\forall v \in S'$, v has at least two edges from S
  // S_r is the reliable node set in S', initially $S_r = \emptyset$
  if $S' \neq \emptyset$ **then**
    for all node $v \in S'$ **do**
      for all edge pair $(e_{u_1,v}, e_{u_2,v})$ from S **do**
        $\bar{h}_{u_1,u_2} = (\bar{h}_{u_1} + \bar{h}_{u_2})/2$
      end for
      Find P_v, set of edge pairs of v satisfying C1 $\wedge$ (C2 $\cup$ C3)
      if $P_v \neq \emptyset$ **then**
        $S_r = S_r \cup \{v\}$
        Choose $(e_{u_1,v}, e_{u_2,v})$ from P_v with min $\bar{h}_{u_1,u_2}$
      else
        Choose $(e_{u_1,v}, e_{u_2,v})$ from S' with min $\bar{h}_{u_1,u_2}$, ties are resolved in flavor of more edges between u_1 and u_2
      end if
      $\bar{h}_v = \bar{h}_{u_1,u_2} + 1$
    end for
    if $S_r \neq \emptyset$ **then**
      Add v in S_r with min $\bar{h}_v$ to S
    else
      Add v in S' with min $\bar{h}_v$ to S
    end if
    ConstructDG(G, u_1, u_2, v);
  else
    Find $S'' \subseteq V - S$ and $\forall v \in S''$, v has one edge $e_{u,v}$ from S
    if $S'' \neq \emptyset$ **then**
      for all node $v \in S''$ **do**
        $\bar{h}_v = \bar{h}_u + 1$
      end for
      Add v to S with min $\bar{h}_v$
      $G_v = (\{u \cup v\}, \{e_{u,v}\})$
      $R_v = R_u \rightarrow G_v$
    else
      return FAIL;
    end if
  end if
end while
return SUCCESS;

Alg 6 ConstructDG (G, u_1, u_2, v)

Let E_δ be the set of edges between u_1, u_2 and from u_1, u_2 to v
if u_1, u_2 satisfy C1 $\wedge$ C2 **then**
  $G_v = G(\{u_1, u_2, v\}, E_\delta)$
  $R_v = R_{u_2} \rightarrow G_v$
else if u_1, u_2 satisfy C1 $\wedge$ C3 **then**
  if u_1 has an edge e from u_2's parent u_2^p in G_{u_2} $\wedge$ u_2 has an edge e' from u_1's parent u_1^p in G_{u_1} **then**
    if $h_{u_2} < h_{u_1}$ **then**
      $G_v = G(\{u_1, u_2, u_2^p, v\}, E_\delta \cup e)$
      $R_v = R_{u_2} \rightarrow G_v$
    else
      $G_v = G(\{u_1, u_2, u_1^p, v\}, E_\delta \cup e')$
      $R_v = R_{u_1} \rightarrow G_v$
    end if
  else if u_2 has an edge e from u_1's parent u_1^p in G_{u_1} **then**
    $G_v = G(\{u_1, u_2, u_1^p, v\}, E_\delta \cup e)$
    $R_v = R_{u_1} \rightarrow G_v$
  else if u_1 has an edge e from u_2's parent u_2^p in G_{u_2} **then**
    $G_v = G(\{u_1, u_2, u_2^p, v\}, E_\delta \cup e)$
    $R_v = R_{u_2} \rightarrow G_v$
  end if
else
  if e_{u_1,u_2} and e_{u_2,u_1} both exist **then**
    $G_v = G(\{u_1, u_2, v\}, E_\delta)$
    $R_v = (h_{u_1} < h_{u_2})\,?\,R_{u_1} \rightarrow G_v : R_{u_2} \rightarrow G_v$
  else if there is neither e_{u_1,u_2} nor e_{u_2,u_1} **then**
    $G_v = (h_{u_1} < h_{u_2})\,?\,G(\{u_1, v\}, \{e_{u_1,v}\}) : G(\{u_2, v\}, \{e_{u_2,v}\})$
    $R_v = (h_{u_1} < h_{u_2})\,?\,R_{u_1} \rightarrow G_v : R_{u_2} \rightarrow G_v$
  else if e_{u_1,u_2} exists **then**
    $G_v = G(\{u_1, u_2, v\}, E_\delta)$
    $R_v = R_{u_1} \rightarrow G_v$
  else
    $G_v = G(\{u_1, u_2, v\}, E_\delta)$
    $R_v = R_{u_2} \rightarrow G_v$
  end if
end if

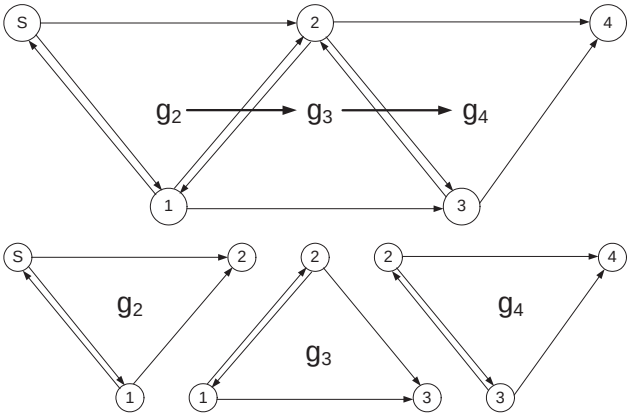

Figure 3.8: An example of the SRDR optimization

$g_2 \rightarrow g_3 \rightarrow g_4$. In node 2, it contains the routing information for both graph g_3 and g_4. It contains edges $2 \rightarrow 3$ and $2 \rightarrow 1$ on g_3 and edges $2 \rightarrow 4$ and $2 \rightarrow 3$ on g_4. When a packet arrives at node 2 with an ordered graph list $g_3 \rightarrow g_4$ in the network layer header (g_2 is removed at node 2), node 2 will take the "shortcut" and try to forward the packet on graph g_4 to node 4. Only if both edges on graph g_4 are broken, node 2 will forward the packet on graph g_3 and try the edge $2 \rightarrow 1$ instead. Under this worse-case scenario, the packet will forwarded to node 4 through $s \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 4$.

3.2.9 Maintaining Reliable Routing Graphs with Network Dynamics

The algorithms presented in the previous subsections construct the reliable routing graphs in ideal scenarios where network devices work properly after joining the network. Although most WirelessHART networks are usually quite stable after deployment, network devices may experience various failures and need to be reset. Wireless links can also be blocked by interference and become temporarily or permanently unavailable. All these scenarios require the Network Manager to recover the routing graphs to maintain the reliability requirements. Furthermore, corresponding adjustments on the communication schedules are also necessary along with these routing graph modifications.

Command	Functionality
Command 779	Report device communication statistics
Command 780	Report neighbor health list
Command 787	Report neighbor signal levels
Command 788	Path down alarm
Command 789	Source route failure alarm
Command 790	Graph route failure alarm

Table 3.3: Summary of network maintenance commands

In WirelessHART networks, network abnormalities and statistics are reported to the Network Manager through a set of network maintenance commands. These commands are summarized in Table 3.3. Command 779 summaries the communication statistics of a specific device; Command 780 and 787 report the signal strengths of a device's neighbors; Command 788, 789 and 790 are triggered once a path failure or routing failure is detected in the network. These commands are carried in normal messages and published to the Network Manager. Based on this information, the Network Manager will update the network topology, adjust the routing graphs and communication schedules if necessary to reach a good balance between the reliability and recovery cost.

Our current heuristics to recover G_B consists of two steps. We first find $G'_B(V'_B, E'_B)$, the sub graph of G_B where all nodes in V'_B are reliable after the topology changes. In the second step, we replace G_B with G'_B and repeat Alg. 1 to incrementally add nodes to G_B. This process repeats until either all the nodes are included in G_B or disconnected nodes are identified. The mechanism to reconstruct G_U is similar to that of G_B. Designing efficient algorithms to reconstruct G_v to each node v is more challenging and will be addressed in our future works.

3.3 Communication Schedule and Channel Management

Typical wireless sensing and control applications take the approach that devices specify their requirements in communication bandwidth and the Network Manager allocates necessary resources such as timeslots, to maintain the periodic sensing-control loop between the Network Manger and devices. In the sensing phase, the devices publish their process data to the Gateway through the uplink graph based on their specific sample rates; In the control phase, the Network Manager generates control messages and sends them back to each individual device on its downlink graph. The Network Manager maintains a global communication schedule for transmitting these process and control data and distributes the sub-schedule to each effected device.

The construction of the communication schedule is subject to several practical constraints in WirelessHART networks:

- The maximum number of concurrent active channels is 16.
- Each device can only be scheduled to TX/RX once in a slot.
- Multiple devices can compete to transmit to the same device simultaneously (in shared timeslot).
- On a multi-hop path, early hops must be scheduled first.
- The practical sample rates are defined as 2^n sec ($-2 \leq n \leq 9$) from 250 ms (2^{-2} sec) to 8 min and 32 sec (2^9 sec).

Our design philosophy for constructing the communication schedule is to spread out the channel usage in the network as much as possible and to apply the Fastest Sample Rate First policy (*FSRF*) to schedule the devices' periodic publishing and control data.

We use the concept of superframe to group a sequence of consecutive timeslots and represent the communication pattern for a given sample rate. We define two types of superframes: data superframe and management superframe. The data superframe is used to

support data transmissions between the devices and the Gateway while the management superframe is used to support exchanging network management messages. The number of data superframes is decided by the number of different sample rates existing in the network. Notice that there can be multiple devices having the same sample rate, thus a data superframe will represent the periodic behavior of multiple devices.

We maintain a global matrix $\mathcal{M}$ to keep track of the current slot/channel usage in the network. Each entry in the matrix, $\mathcal{M}_{i,j}$ represents the slot usage at timeslot i on channel j, and it has four types: unused, exclusive, shared and reserved. An unused entry can be allocated to any pair of devices if there is no communication conflict; An exclusive entry is one occupied by two devices for dedicated communication; Reserved entries are managed by the Gateway or the Network Manager for maintenance purposes; Finally a shared entry allows multiple devices to compete for transmitting to the same device simultaneously. For instance, in our system, we allow 5 simultaneous transmissions on a shared timeslot. We also maintain several other important data structures for constructing the communication schedule. They include one data superframe $\mathcal{F}_i$ per sample rate r_i and a global management superframe $\mathcal{F}_m$. Here we use l_i to denote the length of $\mathcal{F}_i$. For each node v, we maintain a schedule $\mathcal{S}_v$ to record its own slot/channel usage. The length of $\mathcal{M}$ and $\mathcal{S}_v$ are both equal to the maximum length among the existing superframes. These schedules will be distributed to the devices to achieve end-to-end real-time communication.

We present the framework of constructing the data communication schedule in Alg. 7. The construction of the management schedule follows the same approach and is omitted here. In the algorithm, we apply the (*FSRF*) policy in scheduling data transmissions. The construction is based on the reliable graphs we introduced in Section 3.2. For each device v, in its sensing phase, it allocates the primary and retry links along the uplink graph G_U to the Gateway (Line 9 - 10); In the control phase, the Network Manager sends the control messages back and allocates the primary and retry links along the downlink graph G_v (Line 13 - 14). The ScheduleLinks(u, v, G, $\mathcal{F}$, t, o) function is described in Alg. 8. It al-

Alg 7 Constructing Data Communication Schedule

Sort device sample rates in ascending order: $r_1 < r_2 < \ldots < r_k$.
Identify the set of nodes with each sample rate: $N_1, N_2, \ldots, N_k$.
Initialize the schedule for each node as $\emptyset$

for all r_i from r_1 to r_k **do**
    Generate the data superframe $\mathcal{F}_i$
    for all node $v \in N_i$ **do**
        // Schedule primary and retry links for publishing data
        ScheduleLinks(v, g, G_U, $\mathcal{F}_i$, 0, Exclusive);
        ScheduleLinks(v, g, G_U, $\mathcal{F}_i$, $\frac{l_i}{4}$, Shared);

        // Schedule primary and retry links for control data
        ScheduleLinks(g, v, G_v, $\mathcal{F}_i$, $\frac{l_i}{2}$, Exclusive);
        ScheduleLinks(g, v, G_v, $\mathcal{F}_i$, $\frac{3l_i}{4}$, Shared);

        if all link assignments are successfully **then**
            continue;
        else
            // Defer bandwidth request from node v
            return FAIL;
        end if
    end for
end for
return SUCCESS;

locates every link on the paths from u to v on graph G one by one in a depth-first manner. It allocates the earliest available timeslot t_i from t for each link and updates $\mathcal{M}$, $\mathcal{F}$ and each effected node's schedule accordingly. If we cannot find a slot in $[t, l_{\mathcal{F}}]$ to accommodate all the allocations, the Network Manager will defer the bandwidth request from the corresponding device until enough bandwidth resources are available (Line 19 - 20 in Alg. 7).

Notice that a device v is typically multi-hop away from the Gateway, and it has multiple paths to the Gateway due to the property of reliable graph routing. However, if we allocate the required communication bandwidth for device v on each hop along all its paths to the Gateway, most of the allocated links will be wasted because in each end-to-end transmission, only one path will be picked. This will severely degrade the schedulability of the network schedule. To address this problem, as shown in Alg. 8 (Line 17 - 33), when the device has two successors to forward the messages, we reduce the transmission rate between v and each of its successors to half of the original sample rate, and schedule the links on the corresponding superframe $\mathcal{F}'(l_{\mathcal{F}'} = 2 \cdot l_{\mathcal{F}})$. We determine the timeslot offset of these links in $\mathcal{F}'$ to make sure that their combinations will form a communication pattern the same as the original sample rate.

3.4 Performance Evaluation

This section summarizes the major results from our simulations to evaluate the performance of our algorithms. Our simulation model and parameter settings are described in Section 3.4.1. Section 3.4.2 compares our algorithms in constructing reliable routing graphs to traditional approaches. Section 3.4.3 evaluates the performance of our approach for constructing communication schedules. The results show that our approaches can achieve higher routing success rates, better end-to-end communication latency while incurring only modest configuration overheads on devices.

Alg 8 ScheduleLinks(u, v, G, $\mathcal{F}$, t, o)

1: // u and v are the source and destination of the communication
2: // G is the routing graph and $\mathcal{F}$ is the superframe
3: // t is the earliest slot to be allocated and o is the link option
4:
5: Identify data superframe $\mathcal{F}'$ with $l_{\mathcal{F}'} = 2l_{\mathcal{F}}$
6: **for all** node $i \in$ Successor(u) **do**
7: Identify the schedule $\mathcal{S}_u$ and $\mathcal{S}_i$ for node u and i
8: **if** i is the only successor of u **then**
9: Identify the earliest slot from t with a channel c to:
10: Allocate entries $\mathcal{M}_{k \cdot l_{\mathcal{F}} + t_i, c} (k = 0, 1, ...)$ on $\mathcal{M}$
11: Allocate the slots $k \cdot l_{\mathcal{F}} + t_i$ on $\mathcal{S}_u$ and $\mathcal{S}_i$
12: Allocate slot t_i on $\mathcal{F}$
13:
14: **if** All allocations are successful **then**
15: ScheduleLink(i, v, G, $\mathcal{F}$, t_i, o);
16: **end if**
17: **else**
18: **if** i is the first successor **then**
19: Identify the earliest slot from t with a channel c to:
20: Allocate entries $\mathcal{M}_{k \cdot l_{\mathcal{F}'} + t_i, c}$ on $\mathcal{M}$
21: Allocate slots $k \cdot l_{\mathcal{F}'} + t_i$ on $\mathcal{S}_u$ and $\mathcal{S}_i$
22: Allocate slot t_i on $\mathcal{F}'$
23: **else**
24: Identify the earliest slot from t in $\mathcal{M}$ with a channel c to:
25: Allocate entries $\mathcal{M}_{k \cdot l_{\mathcal{F}'} + l_{\mathcal{F}} + t_i, c}$ on $\mathcal{M}$
26: Allocate slots $k \cdot l_{\mathcal{F}'} + l_{\mathcal{F}} + t_i$ on $\mathcal{S}_u$ and $\mathcal{S}_i$
27: Allocate slot $l_{\mathcal{F}} + t_i$ on $\mathcal{F}'$
28: **end if**
29:
30: **if** All allocations are successful **then**
31: ScheduleLink(i, v, G, $\mathcal{F}'$, t_i, o);
32: **end if**
33: **end if**
34: **if** No feasible allocations available **then**
35: return FAIL;
36: **end if**
37: **end for**
38: return SUCCESS;

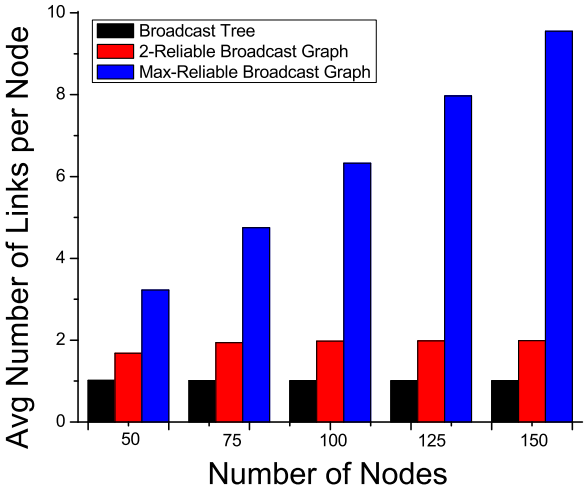

Figure 3.9: Configuration overhead in broadcast graphs

3.4.1 Simulation Model and Parameters

In the simulations, we assume open field, line-of-sight experimental scenarios. The simulation area is fixed at 450 m $\times$ 450 m and the default device communication distance is 100 meters with a 0 dBm transmitter. We assume that there is no edge between a pair of nodes if they are not in each other's communication range. Otherwise, an edge exists with an edge success probability p that is varied from 0.0 to 1.0. The size of the network is varied from 50 to 150 to evaluate the effect of network density on the algorithm's performance. We disable a given portion of links in the network to evaluate the reliability of the constructed routing graphs and this percentage is varied from 0% to 95%.

3.4.2 Performance of Reliable Routing Graphs

We conducted a series of experiments to evaluate the performance of the reliable broadcast graph G_B, reliable uplink graph G_U and reliable downlink graph G_v for each individual node v. Since essentially G_U is the reversed version of G_B, its performance is similar to that

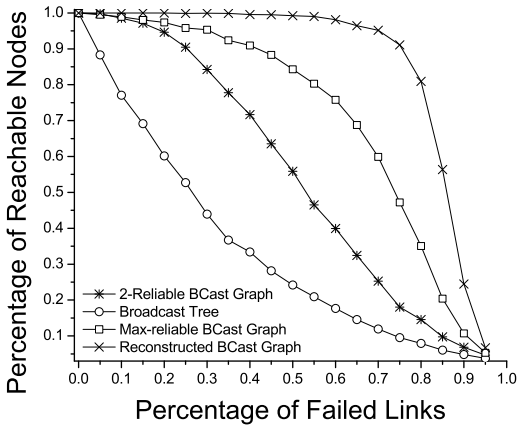

Figure 3.10: Reachability in broadcast graphs

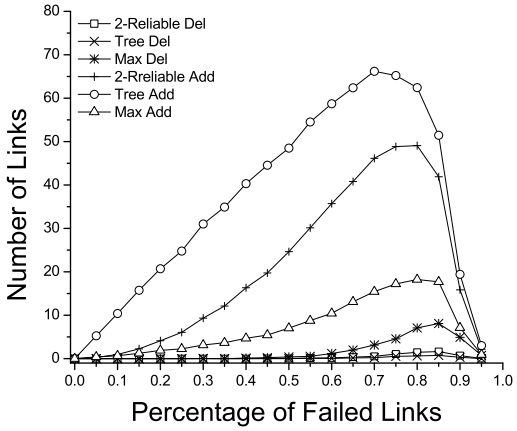

Figure 3.11: Recovery overhead to regain connectivity

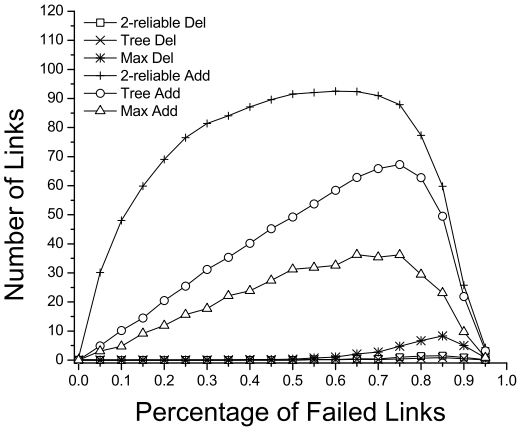

Figure 3.12: Recovery overhead to regain reliability

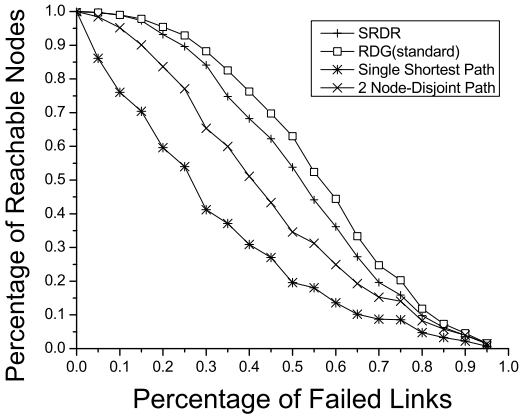

Figure 3.13: Reachability in downlink graph

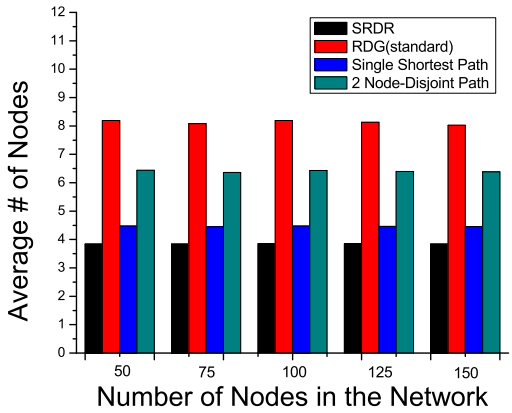

Figure 3.14: Average # of nodes per downlink graph

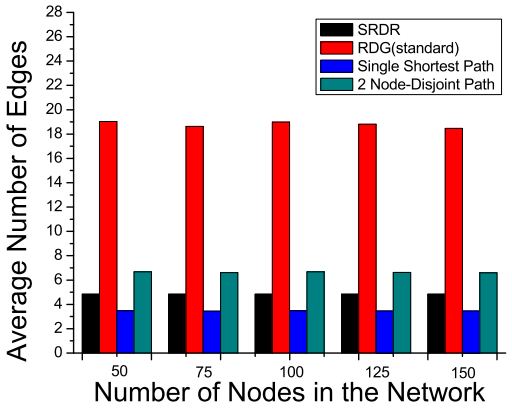

Figure 3.15: Average # of edges per downlink graph

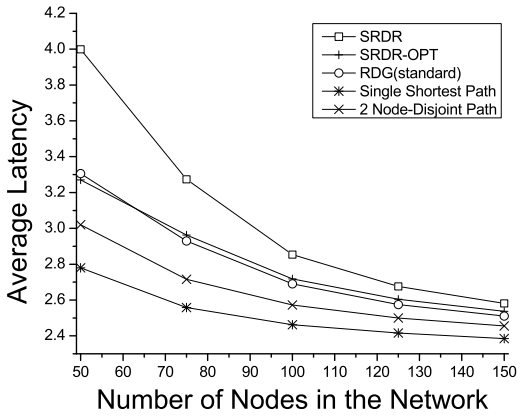

Figure 3.16: Average latency vs. Network size

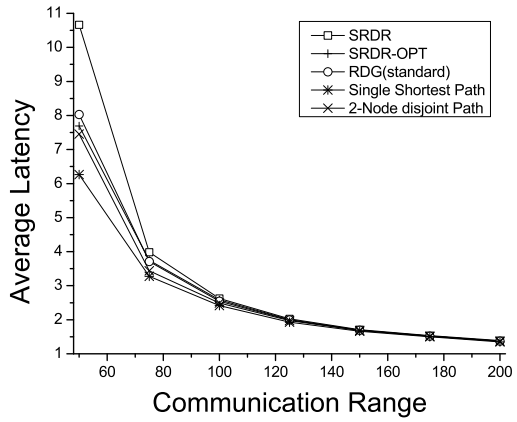

Figure 3.17: Average latency vs. Communication range

of G_B. For this reason, the experiment results of G_U are omitted here.

We compare our approach for constructing G_B with two baseline methods. The first method constructs a single broadcast tree using breadth-first search and the second method generates the max-reliable broadcast graph. In the latter method, when a node is chosen to be added to the broadcast graph, all its incoming edges from the current broadcast graph are also added. Different from this method, our approach only chooses the first two incoming edges of the chosen node with minimum latency, and thus achieve a good balance between the routing reliability and the configuration overhead. In this chapter, the configuration overhead is defined as the average number of links to be configured per node. It is an important performance metric because wireless sensors' memory is limited and configuring large number of links in the network will severely hurt the schedulability of the communication schedule.

The first experiment compares the configuration overhead introduced by these three approaches. In the experiments, we vary the size of the network from 50 to 150 nodes and evaluate its impact. Figure 3.9 summarizes our results. As expected, we observe that the configuration overhead of the max-reliable approach is much higher than the other two and it increases linearly along with the increase of the network density. On the other hand, the overhead in our approach and the broadcast tree solution is much low and stable. The overhead in our approach is always below 2 links per node, and it is closer to the performance of the broadcast tree when the network density is low. This observation is mainly because when the network density is low, it is difficult for many nodes to find two parents in the network thus has only one link in the broadcast graph.

In the second experiment, we first construct the broadcast graphs based on these three approaches with 100 nodes in the network. We then gradually increase the percentage of failed links in the network from 0% to 95%. We measure the reliability of these three approaches and apply the recovery mechanisms we discussed in Section 3.2.9 on them. We compare their recovery overhead in terms of number of changed links. Figure 3.10 shows

that along with the increased percentage of failed links in the network, the reliability of the broadcast tree drops quickly and when half of the links die, only around 25% nodes are reachable from the Gateway. Our approach performs much better. With the same percentage of failed links, around 55% of nodes are still connected. Among all three approaches, the max-reliable broadcast graph has the best performance as a tradeoff of its poor scalability and much higher configuration overhead. In Figure 3.10, we also show a curve of the reachability for the broadcast graphs after the recovery. As the recovery mechanisms are all based on the same underlying network topology, all three approaches have the same reachability after reconstruction. This in turn verifies the correctness of our recovery mechanisms.

Figure 3.11 and Figure 3.12 compare the recovery overhead among these approaches. Figure 3.11 shows the overhead to resume the connectivity of the broadcast graphs while Figure 3.12 further shows the overhead to recover their reliability properties. We observe from Figure 3.11 that the broadcast tree always has the heaviest recovery overhead while the max-reliable broadcast tree has the minimum because of its best reliability. The performance of our approach sits between them. However, Figure 3.12 shows that to recover the reliability property, our approach needs to add more links than the other two alternatives. The reason is the broadcast tree has no reliability requirement while the max-reliable approach has already added most of the links in the construction stage thus its recovery overhead is relatively smaller.

In the third experiment, we evaluate the performance of the two proposed approaches for constructing reliable downlink graphs, the standard approach as defined in WirelessHART standard RDG(standard) and the sequential reliable downlink routing approach (SRDR). we compare them with two baseline methods. The first method finds a single shortest path from the Gateway to the destination, while the second one constructs a two node-disjoint path and can tolerate one link or node failure. Figure 3.13 summarizes the comparison of the routing reliability among these four approaches. It clearly shows that the

single path approach always has the worst performance. On the other hand, RDG(standard) maintains the best reliability and always outperforms the two node-disjoint path method more than 30%. SRDR is around 8% worse than RDG(standard) in routing reliability. This is because the downlink graphs constructed under RDG(standard) have more redundant links. As a tradeoff, as shown in Figure 3.14 and Figure 3.15, RDG(standard) introduces a much higher configuration overhead. The average number of nodes in the constructed graphs is 2 times and 1.2 times larger than that of the single shortest path approach and two node-disjoint path approach respectively. Furthermore, as each node under RDG(standard) has two outgoing edges, the average number of links in the constructed graphs is even higher. As shown in Figure 3.15, it is around 5.5 times and 2.8 times larger than that of the single shortest path approach and two node-disjoint path approach respectively. However, SRDR only introduces very limited configuration overhead because it only constructs local graphs and these local graphs can be further reused for assembling the downlink routes to different destinations. Its average number of nodes is the lowest among all the four approaches and its average number of links is only slightly higher that of the single shortest path approach and around 33% lower than the two node-disjoint path approach. In sum, SRDR achieves a good balance between high routing reliability and low configuration overhead.

We also evaluate the performance of the optimization mechanism SRDR-OPT which is proposed in Section 3.2.8, and measure its improvement on average latency in two different scenarios. In the first scenario, we fix the devices' communication range at 100m and increase the number of nodes in the network from 50 to 150. The results is shown in Figure 3.16. We observe that SRDR has a much higher average latency compared with RDG(standard). This is because when constraint C2 is satisfied, SRDR chooses the node with larger latency as its parent in constructing downlink graph while RDG(standard) take both and its latency is calculated as their average plus one. The performance of SRDR-OPT is similar to RDG(standard) because the shortcuts are taken in the optimization. Obviously, the single shortest path approach always has the lowest latency. In the second scenario, we

fix the number of nodes in the network at 150 and vary the communication range of the devices from 50m to 200m. As shown in Figure 3.17, the average latencies of all the four approaches decrease with the increase of the communication range, and consistent with the observations in the first scenario, SRDR has a great improvement on the average latency when the optimization mechanism is applied.

3.4.3 Construction of Communication Schedules

Our approach for constructing the communication schedule has two unique features. First, we split the traffic from a device among all its successors by reducing the bandwidth requirement on each successor. The communication schedules on the successors are carefully designed so that their combination has the same patten as the original device. Second, we use the concept of shared timeslot to allow multiple devices to compete for communicating with the same device simultaneously. This is especially useful for the links that are allocated for retry purpose and it can significantly improve the network throughput.

In this section, we evaluate the performance of these two features by comparing our approach with three baseline methods. The basic methods either lack one of the features or both of them. For simplicity, we only show our experimental results on scheduling process data from devices to the Gateway on the uplink graph. Scheduling control data on the other direction is similar, and thus is omitted here. Two performance metrics are defined for this experiment. The first metric is the scheduling success ratio which measures the percentage of nodes that can successfully allocate the required bandwidth along its paths to the Gateway; The second metric is the network utilization which measures the percentage of entries in matrix $\mathcal{M}$ that are already allocated for communication. Our results are summarized in Figure 3.18 and Figure 3.19 respectively.

In Figure 3.18, we compare the scheduling success ratio by deploying 50 nodes in the network and varying the device sample rate from 250 ms to 4 min and 16 sec (each device has the same sample rate). We observe that by halving the bandwidth requirement on

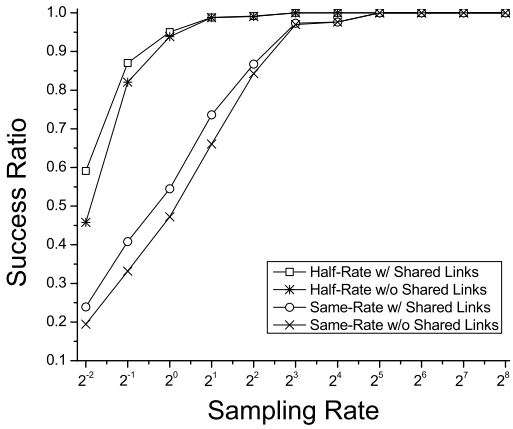

Figure 3.18: Success ratio vs. Sample rate

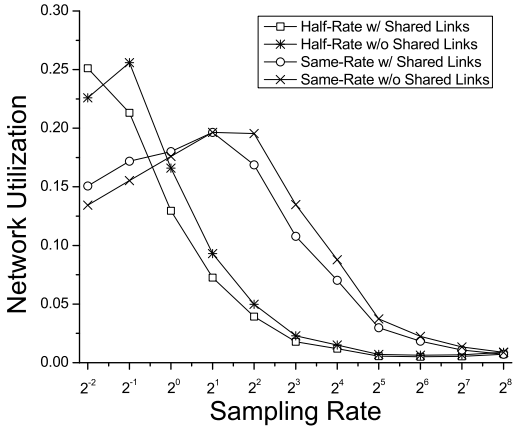

Figure 3.19: Network util. vs. Sample rate

a device's successors (if it has two successors), the success ratio can be greatly improved. The improvement is more than 25% when the sample rate is 2 sec and is even higher when the sampling is faster. Figure 3.18 also shows that by applying the shared timeslot, the success ratio can be increased by 5% and this improvement is consistently shown in our experiment results until the sample rate is low enough that the scheduling success ratio approaches 100%. Figure 3.19 shows that when the approaches have a similar scheduling success ratio, our approach has a much lower network utilization, and this will further help include more devices into the network. When the sample rate is fast, our approach has a higher network utilization because in these scenarios, the success ratio for other approaches is so poor that a very limited number of devices can successfully allocate their required bandwidth along its path to the Gateway.

3.5 Summary

In this chapter, we study the problem of how to achieve reliable and real-time services in CPS subsystems. Taking WirelessHART network as an example, we abstract the reliability

requirements in typical wireless sensing and control applications and present the algorithms for constructing three types of reliable routing graphs for different communication purposes. Based on these routing graphs, we describe how we construct the communication schedule in the network and highlight our approach's unique features. Extensive simulations are conducted to evaluate the performance of the proposed algorithms.

As ongoing and future work, we shall continue to look for more efficient approaches for constructing routing graphs and communication schedules to minimize the communication latency and maximize the power saving in wireless sensing and control networks. We will also study their corresponding recovery mechanisms in presence of network dynamics.

Chapter 4

System Design, Implementation and Deployment

In this chapter, we will describe the design details and implementation of our complete wireless real-time communication system. Figure 4.1 depicts the architecture of the system which has five major components: the devices which form a wireless real-time mesh network, Access Point, Network Manager, Gateway and Host applications. These components are shown in Figure 4.2, and their design details will be presented in the following sections respectively. We will also describe two important tools we have built: a compliance verification environment called Wi-HTest to facilitate device compliance assessment, and a network simulator to help design network management techniques, exercise them on specified network topologies and evaluate their performance. The system and tools together provide an ideal platform for wireless real-time sensing and control networks. The platform can serve as a fundamental communication infrastructure for supporting reliable and real-time services in a variety of cyber-physical systems. In the last part of this chapter, we will describe our effort in deploying the system in a variety of environments including university labs, industrial plants and power houses. We aim at building stable testbeds in these testing environments, collecting long-term experimental data and evaluating the performance of ex-

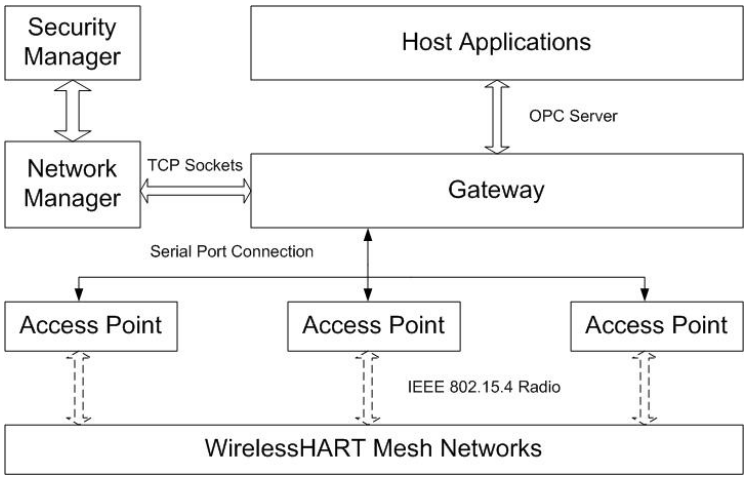

Figure 4.1: Architecture of the complete WirelessHART communication system

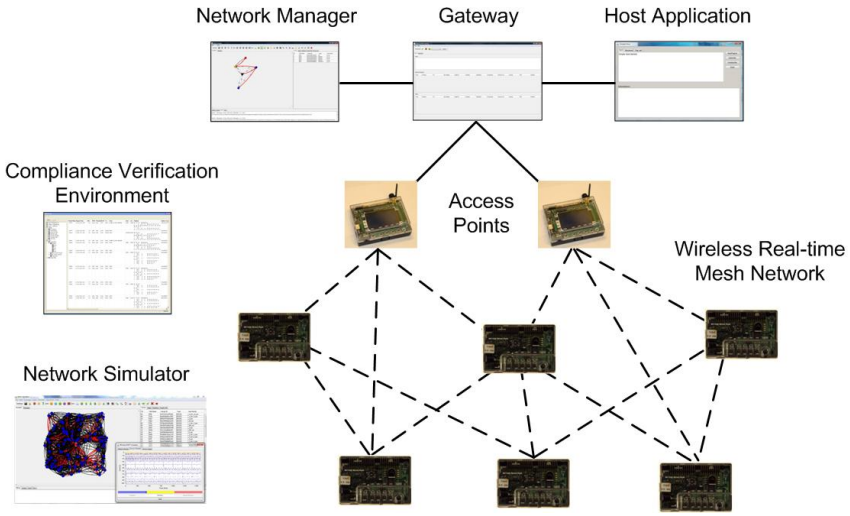

Figure 4.2: The major components in the system

isting and to be proposed network management techniques. The findings from the testbeds will help design more efficient routing and scheduling algorithms in achieving reliable and real-time services in mission-critical cyber-physical systems.

4.1 Hardware Platforms

There are many commercial-off-the-shelf (COTS) IEEE 802.15.4-based hardware chips on the market, however most of them cater to ZigBee applications. The processor power and memory space are designed to be small to consume less power. In other words, while these chips are powerful enough to cover ZigBee applications, they are limited to support a fully fledged WirelessHART stack. A full-blown WirelessHART stack needs more memory space for supporting larger data tables and more complicated functionalities, more powerful processor and hardware accelerator to perform encryption and decryption with hard deadline constraints and more precise clock to achieve network-wide synchronization. Keeping all these constraints in mind, by early 2011, the MC1322 System-in-Package (SIP) chip from FreeScale [6] is the only hardware platform that can satisfy our requirements. For this reason, we choose the Freescale MC1322x evaluation kit [1] as our hardware platform for developing our communication stack. MC1322x has the following features:

- 2.4 GHz wireless nodes compatible with the IEEE 802.15.4 standard
- Advanced AES hardware engine and hardware acceleration for 802.15.4 applications
- 32-bit ARM7TDMI-S CPU core with programmable performance up to 26 MHz
- Extensive on-board memory resources: 128 Kbyte serial FLASH memory, 96 Kbyte SRAM and 80 Kbyte ROM
- 22mA typical RX current draw, 29mA typical TX current draw
- Extensive sleep mode control and variation

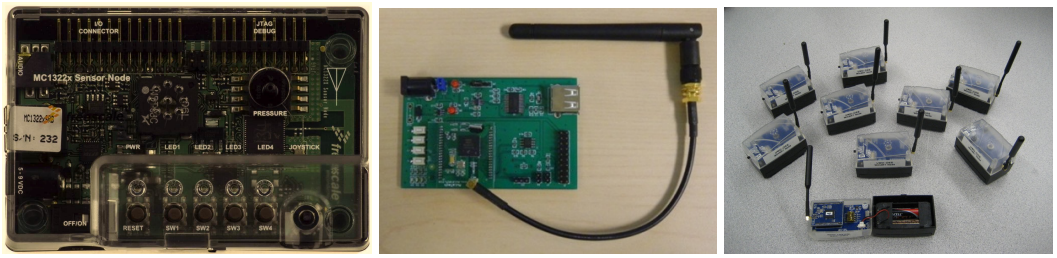

Figure 4.3: Evolution of our hardware platforms for WirelessHART embedded device

To further reduce the energy consumption on our hardware platform, we collaborated with the Control Instrumentation & Electrical Systems (CIES) Lab in the University of Western Ontario. We made significant effort to redesign the mother board by removing unnecessary circuits/unused sensors, and reducing the board dimensions. Figure 4.3 gives an overview of the Freescale 1322x-SRB (Sensor Reference Board) and our generation I and generation II hardware platforms.

4.2 Access Point Design

The Access Point is a bridge between the real-time mesh network and the Gateway. There could be multiple Access Points attached to the Gateway providing load balancing and reliable graph routing. Our Access Point is implemented on the same hardware platform as our device. Each Access Point goes through the same join procedure as a normal device as described in Section 2.6 to authenticate itself and establish a secure connection with the Network Manager. As shown in Figure 4.4, the communication stack on the Access Point is extended from that of the device by adding an extra UART module for serial port communication. The messages received from the devices in the mesh network will be forwarded to the UART module by the network layer and sent to the Gateway. In the other direction, the messages from Gateway/Network Manager will be conveyed through the UART module and put into the network layer queue in the Access Point for dispatching.

If the network system includes multiple Access Points, they must be strictly syn-

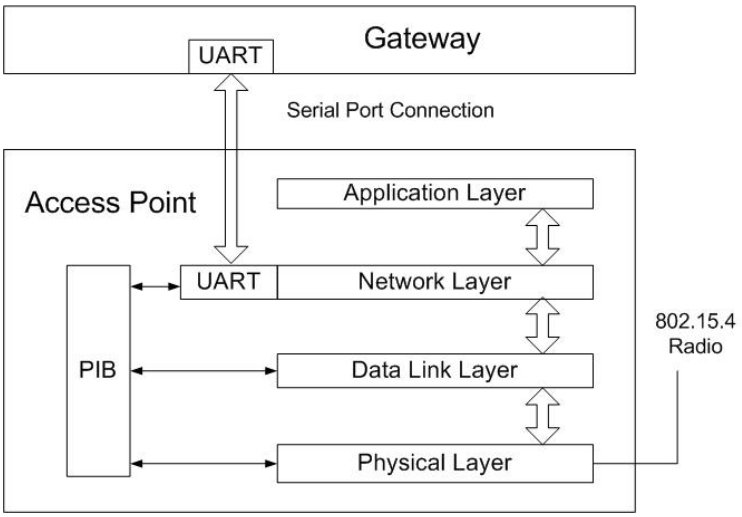

Figure 4.4: The architecture of the Access Point

chronized. In our design, except the designated time source, all other Access Points will be instructed by the Network Manager to scan the physical channels in the same way as when a normal node joins the network. A normal node will send out join request message to a neighbor after synchronization. These Access Points directly send the join requests to the Network Manager through the Gateway. Afterwards the Network Manager configures them just like it configures the original time source.

4.3 Gateway and Host Application Interface

The Gateway works as a server and is responsible for communicating with the Network Manager, processing the requests from the Host applications, collecting and caching measurements from all devices in the network. The Gateway provides abundant APIs for the Host applications to interact with the sensors and actuators in the network for providing sensing and control services. The architecture of the Gateway is illustrated in Figure 4.5 and its major components are summarized in the following.

Physical Connections: The Gateway implements an abstract communication layer and pro-

vides serial port connection (RS232 or RS485) to each attached Access Point. It talks with the Network Manager through a TCP socket connection for message exchange thus they can be located in different physical locations. The Gateway also provides abundant Host Interfaces to backbone networks to receive the queries and send back responses. Frequently used Interface commands are summarized in Table 4.1.

Real-time Database and Query Processor: Two essential components of the Gateway are a real-time database and a query processor. The database provides data caching for burst mode, event notification, and common HART command responses. The query processor processes the queries from Host applications. If the requested data are already cached and still valid, they are returned immediately to the Host applications. This reduces network traffic and improves the Host application's responsiveness. Otherwise, the query processor will generate the request messages and send them to the corresponding devices. The return response data are cached in the Gateway and sent back to the Host applications.

Time Source: The Gateway maintains a time source module for maintaining network-wide time synchronization. It will notify all the devices in the network and let them synchronize with the Gateway. If the Gateway is not implemented in real-time embedded platforms, the actual time source is a designated Access Point instead of the Gateway. The time source will periodically update the accurate time to the Gateway and Network Manager.

Control Function Block: Different from traditional approaches where the control module runs in the Hosts function block application, we are experimenting the Control-in-Gateway approach in our system by enhancing our Gateway with a function block application layer to allow configuration and execution of the control modules. Benefits of this approach include: (1) a deterministic schedule is established for all communications by the Network Manager; (2) function block execution may be fully synchronized with IO communication; and (3) control strategy may be fully backed up using redundant Gateways.

User Interface: We designed a user-friendly graphic interface to present the users and network administrators a simple and clear summary of the exchanged messages among the

ID	Name	Definition
1	**ReadTagList** ()	Read the list of devices in the network
2	**Subscribe** (tag, update rate, burst command number)	Instruct the device to publish data to the host. The data is also saved in the Gateway cache
3	**Unsubscribe** (tag)	Stop publishing
4	**Read** (tag)	Read data from the Gateway cache
5	**TagList** (tag list)	Report the device list
6	**Data** (tag, data value)	Forward the device data to Host
7	**Write** (tag, data value)	Write data to the device
8	**HARTCmd** (tag, command number, command content)	Send a HART command to designated device

Table 4.1: Gateway interface for Host applications

Network Manager, the Gateway and each device in the network. It also visualizes all the queries received from Host applications and the corresponding responses.

4.4 Network Manager Design

The core of a WirelessHART mesh network is the Network Manager. It is responsible for authenticating the devices, forming the network, allocating network resources and scheduling process data transmissions. We have described the detailed algorithmic issues in Chapter 3 for generating routing graphs and constructing communication schedules to achieve reliable and real-time communication. In this section, we describe our design and implementation of the Network Manager and how we integrate our network management techniques into it. Figure 4.6 shows the architecture of the Network Manager which has the following four major components:

Command Processor: The application layer of the WirelessHART standard is command-oriented. The WirelessHART devices and the Network Manager interact by exchanging

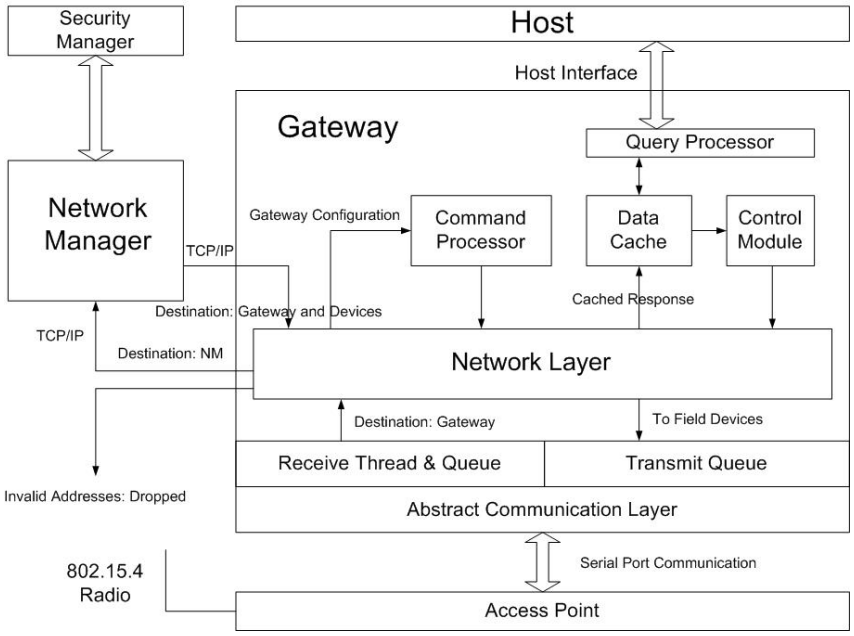

Figure 4.5: The architecture of the Gateway

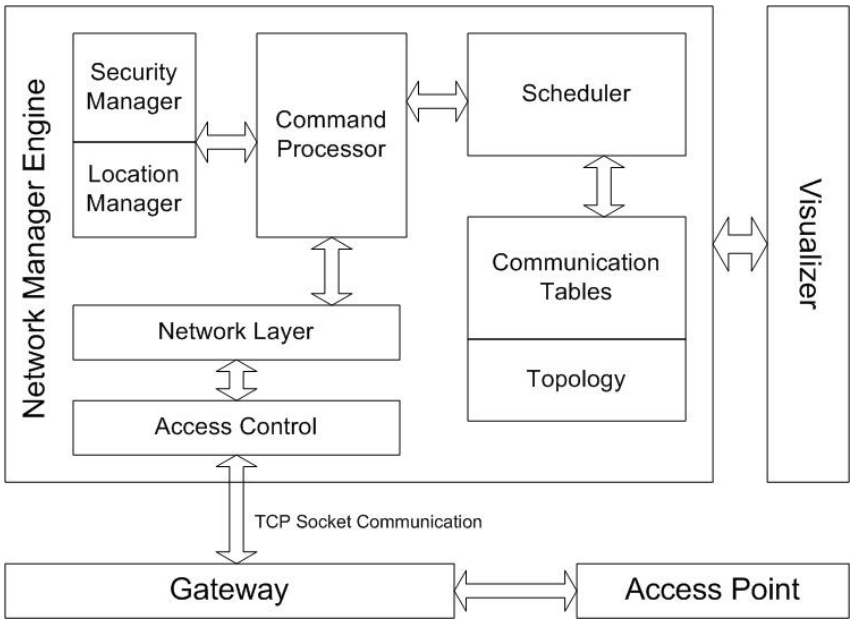

Figure 4.6: The architecture of the Network Manager

command requests and command responses. The command processor in the Network Manager processes the commands from the devices, updates the network topology and device health information. It invokes network management algorithms if necessary to reconstruct the routing graphs and communication schedules to maintain good network performance.

Network Topology and Communication Tables: In the Network Manager, the network topology is maintained in a directed graph structure. All the algorithms for constructing routing graphs and allocating network resources are conducted on the graph and the results are maintained in a set of communication tables, and distributed to corresponding devices. For the details of the communication tables to be maintained in the Network Manager, please refer to Figure 2.8.

Security Manager and Access Control: WirelessHART is a secure wireless communication protocol and it provides encryption and authentication in both the data link layer and network layer. The main task of the security manager is to manage various key information for the devices. It takes charge of the device join authentication and updates the key information in the network periodically for protection purpose. The access control module maintains a list of pre-approved devices together with their valid join keys. Only the devices on the list can be admitted into the network by providing the correct join keys.

Visualizer: Our visualizer is implemented based on the JUNG library [11]. It provides the user a straight-forward way to observe the network topology, the routing graphs, the device communication schedules, and the exchanged messages. Any update on them will also be reflected in the visualizer in real-time. With the visualizer, users can identify problematic network topology and bottlenecks limiting network throughput and perform appropriate adjustments. As shown in Figure 4.6, the visualizer is a stand-alone software which exchanges messages with the Network Manager engine through pre-defined APIs. This makes the porting work to resource-limited embedded platform straightforward.

4.5 Tool: Network Simulator

To evaluate the performance of the network management techniques, especially for achieving reliable and real-time services in large-scale networks in different operating environments, we further enhanced the Network Manager to be a generic network simulator. The simulator allows the users to specify the network topologies either through reading in topology files collected from real-world deployment or from random generation by injecting arbitrary node and/or edge failure and interference from other networks in the surrounding environments. The simulator provides a powerful tool for the users to visualize the network topology, design their routing and scheduling algorithms, exercise them on specified topologies and evaluate their performance. Figure 4.7 and Figure 4.8 present the screen captures of a network simulation with one Gateway, two Access Points and 100 devices in the network. Figure 4.7 shows the network topology, the global reliable broadcast graph (red lines) and the communication schedule for each device based on their bandwidth requirements. Figure 4.8 shows the global reliable uplink graph (blue lines) constructed based on the same network topology as in Figure 4.7 and the bandwidth utilization for each device in the network. This can help the users and network administrators monitor the network, identify abnormality in the network and take corresponding actions to improve the network performance.

4.6 Tool: Wi-HTest Compliance Verification System

To ensure the compliance of the developed communication stack with the WirelessHART communication protocol and the adherence to its strict timing requirements, we collaborated with HART Foundation and developed a complete compliance verification environment for WirelessHART devices. This environment includes a test suite called Wi-HTest for exercising the Device under Test (DUT) with specified testing scripts, a specific sniffer called Wi-Analys for real-time monitoring of the WirelessHART network traffic and a post

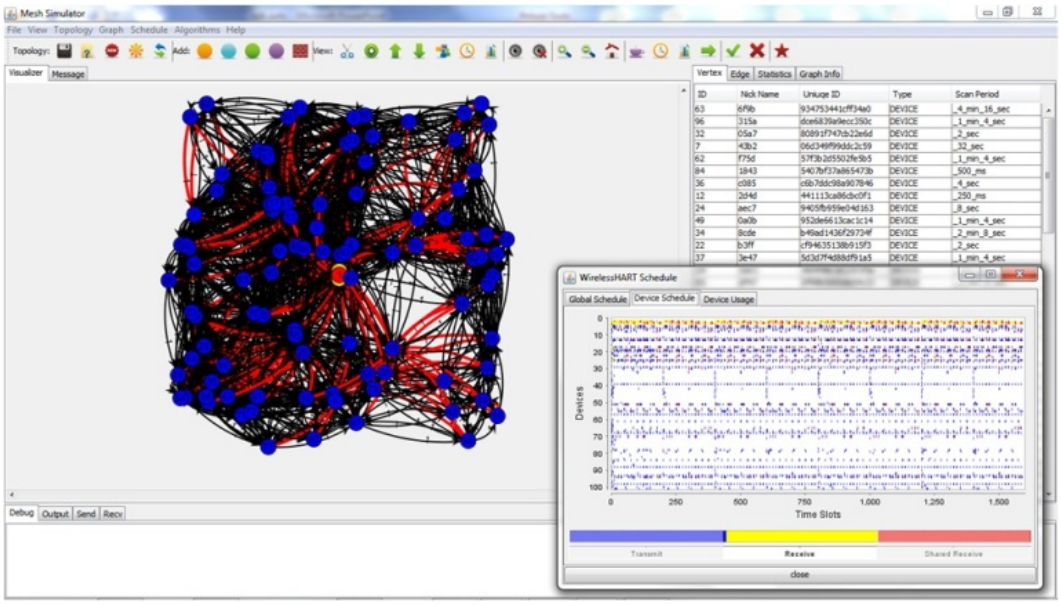

Figure 4.7: A simulation of 100 devices with original network topology, broadcast routing graph and device communication schedule

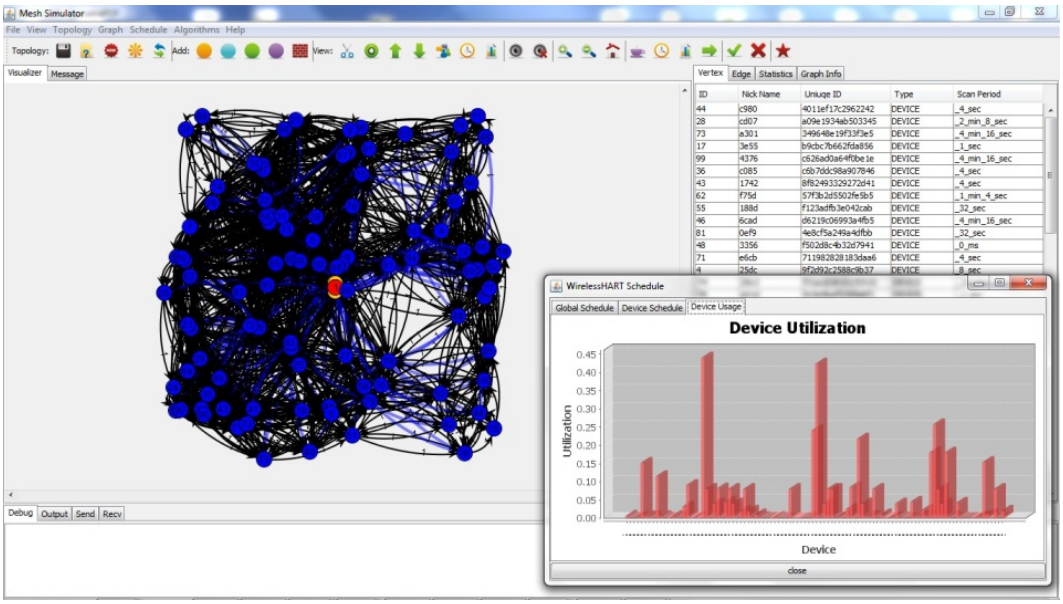

Figure 4.8: A simulation of 100 devices with original network topology, broadcast routing graph and device bandwidth utilization

processing suite for analyzing the packets captured by Wi-Analys and generating the final compliance report.

Wi-HTest aims at automating the execution of test cases defined in the WirelessHART test specification. More specifically, Wi-HTest provides the stimulus (good and bad) necessary to exercise operations of the DUT. The test cases for verifying the compliance of WirelessHART devices can be roughly classified into three different test scenarios, the device join process, MAC layer data communication and network layer data communication. In the join process, Wi-HTest coordinates with the DUT through a sequence of message exchanges and verifies whether the DUT can join the WirelessHART network successfully. In the MAC layer communication tests, Wi-HTest transmits either correct data packets or manipulate the packets by injecting fault data. By evaluating DUTs corresponding response including the precise timing information, the DUTs MAC layer compliance can be assessed. Wi-HTest conducts the network layer communication tests by introducing the concept of virtual network and virtual devices. It simulates a WirelessHART network by adding necessary virtual devices and configuring the communication schedules between the DUT and virtual devices or among virtual devices. End-to-end communications are executed by Wi-HTest to evaluate the DUTs network layer compliance by comparing its practical behaviors with the standard ones according to the WirelessHART specification. Wi-HTest consists of two components: the Wi-HTest Host and an RF Interface. The Wi-HTest Host is responsible for overall control and execution of the input test scripts. The RF Interface is a compact WirelessHART stack. It is responsible for low-level, time-critical communication to and from the DUT using its onboard wireless transceiver. Responses from DUT are forwarded back to the Wi-HTest Host and are also captured by Wi-Analys for post processing. The actual hardware in the Wi-HTest test suite is shown in the left side of Figure 4.9.

Wi-Analys is a multi-channel sniffer designed to capture all 802.15.4 packets in the 2.4 GHz frequency range but focuses on those from WirelessHART devices. The receiver has the capability of capturing data on 16 WirelessHART channels simultaneously and at

Figure 4.9: Wi-HTest test suite and Wi-Analys sniffers

a speed of up to 1000 messages per second. As shown on the right side of Figure 4.9, Wi-Analys consists of a radio receiver box and a software suite running on a workstation. The receiver box is connected to the workstation via the USB cable. The software suite logs all captured WirelessHART messages on all channels. Wi-Analys also displays captured messages in an organized manner, either online or redisplaying a captured log file. The messages are interpreted and the fields in the messages, from physical layer fields all the way up to the application layer fields, could be displayed in columns. Further, intelligence is built-in to decipher the messages so that enciphered fields could be shown in plain text.

The post process suite judges the successfulness of the compliance test. For each test case, a post process program reads the log file and analyzes it. Depending on the purpose of each test case, it will check the sequence of the messages the DUT transmitted, the transmission time points, the relationship of the messages, the content of the messages, etc. If all satisfy the standard, the test case is passed. Otherwise the place where the standard is violated will be reported.

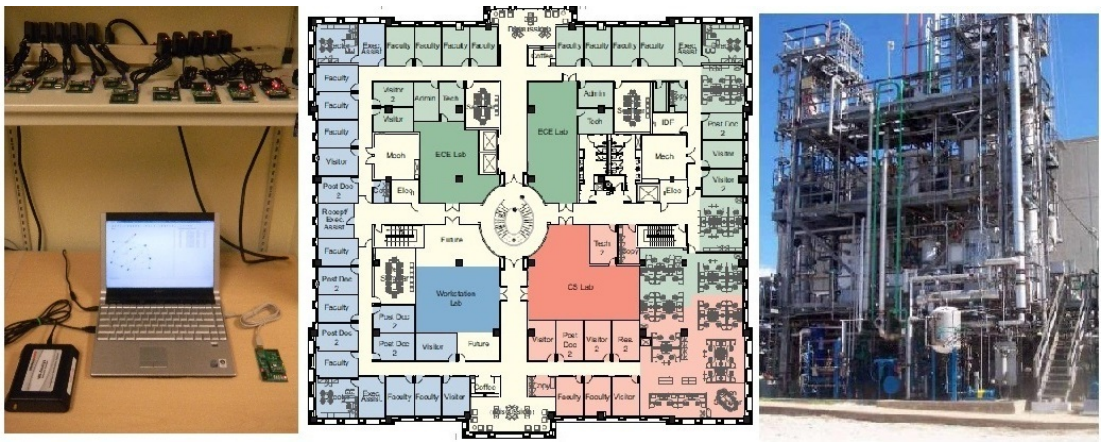

Figure 4.10: An Overview of the WirelessHART communication system and its deployment in UT ACES 5th floor and UT Pickle research center

4.7 Deployment in Different Environments

Putting all the components together, we have built a middle-size WirelessHART testbed which is shown on the left side of Figure 4.10. The system consists of the Network Manager, Gateway, one Access Point and 20 devices. In our tests, all the devices were configured to publish their sensor data (some through multi-hop) to the Gateway every 4 seconds. In our test period of two weeks, the system had been running perfectly with no single packet loss was detected. As the ongoing work, we are adding more Access Points and devices into the testbed to form a larger network in the scale of a hundred nodes. We are deploying the network in typical office environment (UT ACES 5th floor as shown in the middle of Figure 4.10) and industrial plant (Petroleum Engineering Department in UT Pickle Research Center as shown on the right side of Figure 4.10). Measurements will be collected from these environments to evaluate the performance of the network management techniques proposed in this section.

Chapter 5

Interconnecting Heterogeneous Cyber-Physical Subsystems

In Chapter 2 and Chapter 3, we describe the real-time wireless communication protocol we designed and developed for cyber-physical systems and the network management techniques for achieving reliable and real-time intra-CPS services. A large-scale cyber-physical system, however usually involves many heterogeneous cyber-physical subsystems which adopt different physical and data link layer technologies and are distributed in different locations. Connecting heterogeneous embedded devices in different CPS subsystems and achieving reliable and real-time inter-CPS services is a big challenge.

One straightforward solution is to connect the Gateway of each subsystem to the Internet and let the Gateways take the role of protocol adaptors. Although this approach may not require modifications on the devices, it pushes all the complexity to the Gateway. Significant engineering effort is needed to enhance the Gateway of each subsystem to understand other existing protocols in the system. The situation becomes even worse if the Gateway works as a blackbox from a third-party vendor and cannot be easily accessed, or more subsystems need to be integrated in the system in the future. To overcome these problems, an alternative is to only enhance the Gateway with standard IP routing functionality.

In the meanwhile, if certain devices are involved in the inter-CPS services, we will enhance them with an IP adaptation layer and provide end users the standard socket APIs for application development on top of it. In such a way, instead of a protocol adaptor, the Gateway in each subsystem simply works as a router, and all IP-enabled devices in different subsystems are connected through IP and thus can establish direct end-to-end communication. This solution has the following advantages:

- **Scalable Service:** Since the devices are IP-enabled, the Gateway only needs to take the role of the router and forward the IP packets to the destination. Only the source and destination devices need to understand the application protocol of the service. The Gateway can remain unchanged when new services are established between devices in different cyber-physical subsystems. Only the application layers of the devices who are involved in the service need enhancement.

- **Easier Application Development:** The IP-enabled device will provide standard socket APIs for the end users to develop applications. It does not require any specific knowledge about the physical and data link layer of the cyber-physical subsystem which the device is residing in. The developed applications can be easily ported to other IP-enabled devices which may be based on different data link layer technologies.

- **Incremental Deployment Support:** The IP-enabled devices and IP-based services can be added into the cyber-physical subsystems incrementally. Existing devices and services in the cyber-physical subsystems do not need any change.

Enabling IP functions in resource- and bandwidth-limited wireless embedded devices and interconnecting heterogeneous cyber-physical subsystems together to guarantee required quality of service (QoS) has many challenges to address. These challenges include how to map larger IP datagrams to the services provided by the cyber-physical subsystems which usually only support much smaller frame size; how to establish secure end-to-end

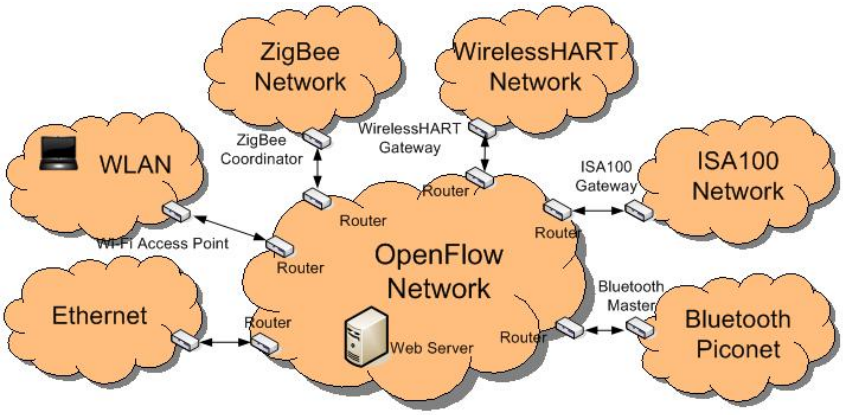

Figure 5.1: Infrastructure of embedded Internet

communication and guarantee QoS between end devices from different subsystems; what routing protocol to apply in this infrastructure and how to simplify existing application protocols in the Internet to make them suitable for the Embedded world. In the following sections of this chapter, taking WirelessHART network as an example, we will present our design details of the networking infrastructure for large-scale heterogeneous cyber-physical systems. We will also describe our implementation of a prototype system which integrates WirelessHART subsystem into the Internet and support web-based monitoring and control services.

5.1 Networking Infrastructure

Figure 5.1 gives an overview of the networking infrastructure we are building for distributed heterogeneous cyber-physical systems. CPS subsystems based on different wireless technologies including WirelessHART, Wi-Fi, ISA100, ZigBee and Bluetooth are connected together through their edge routers (Gateways) who are responsible for routing IP traffic in and out of the subsystems. IP-enabled embedded devices in the subsystems are each assigned an IPv6 address and share a common IPv6 address prefix. To achieve IP functionality on embedded devices and routing functionality on Gateways, enhancements are needed on

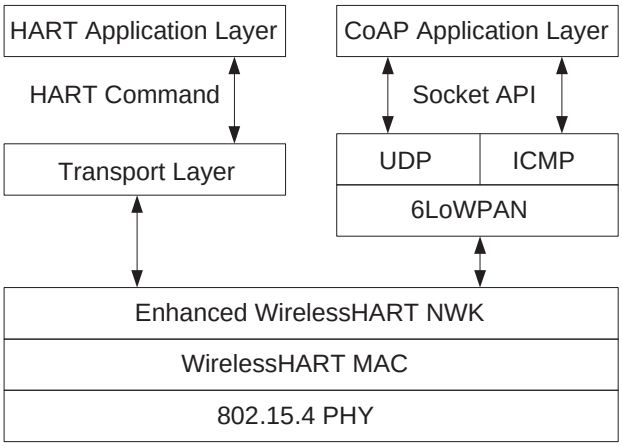

Figure 5.2: Design of the enhanced WirelessHART communication stack

both the device stacks and Gateways. To guarantee the QoS and achieve real-time performance for inter-CPS communication, we also propose to use OpenFlow [58] network to interconnect all the Gateways. The design details of our networking infrastructure will be presented in the following of this section.

5.1.1 Enhancement on Communication Stack

Taken WirelessHART as an example, the architecture of the enhanced communication stack with IP function support is presented in Figure 5.2. With the existing transport layer and application layer unchanged, we designed and built a slim IP stack on top of the WirelessHART network layer. IP packets will be compressed, fragmented and wrapped as the payload of normal WirelessHART packets and sent to Gateway where reassembling and decompression will be executed and then forward to the destination. These two upper layer stacks work together simultaneously so that the enhanced stack can support both normal WirelessHART traffic and IP traffic. Figure 5.3 illustrates the format of the IP-enabled WirelessHART packet. A reserved bit in the control byte of the network layer header (0 by default) is used to differentiate the normal WirelessHART messages from the IP messages

so that they can be dispatched to different upper layer stacks for proper processing. The IP stack includes a 6LoWPAN adaptation layer, a UDP/ICMP transport layer and a CoAP application layer [1].

The 6LoWPAN adaptation layer is necessary because with IP function support, IPv6 packets will be routed in and out of the cyber-physical subsystems and the minimum frame size for standard IPv6 packet is 1280 bytes. Since most cyber-physical subsystems are embedded sensor and actuator networks, the radio technologies adopted there usually have limited bandwidth and small frame size, thus an adaptation is needed. The 6LoWPAN adaptation layer provides encapsulation, header compression and fragmentation mechanisms that allow IPv6 packets to be sent to and received from over low-bandwidth wireless embedded networks.

Considering the limited processor power and small memory in the embedded device, we apply the Constrained Application Protocol (CoAP) [3] in our infrastructure. CoAP provides a method/response interaction model between application end-points, supports built-in resource discovery, and includes key web concepts such as URIs and content-types. CoAP easily translates to HTTP for integration with the web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

5.1.2 Enhancement on Gateway

Figure 5.4 presents the architecture of the enhanced WirelessHART Gateway to function as an edge router between the devices in the embedded network and the Servers/Hosts in Internet or other cyber-physical subsystems. To achieve this, we enhance the Gateway with the following modules: IPv4/6 routing, 6LoWPAN adaptation (compression/decompression, fragmentation/reassembling), 6to4 IP tunneling and Host-to-Host IPSec configuration. As mentioned in Section 5.1.1, the 6LoWPAN adaptation layer enables regular IP messages be

[1] I would like to thank AwiaTech corporation for allowing using the CoAP and 6LoWPAN software for performance evaluation.

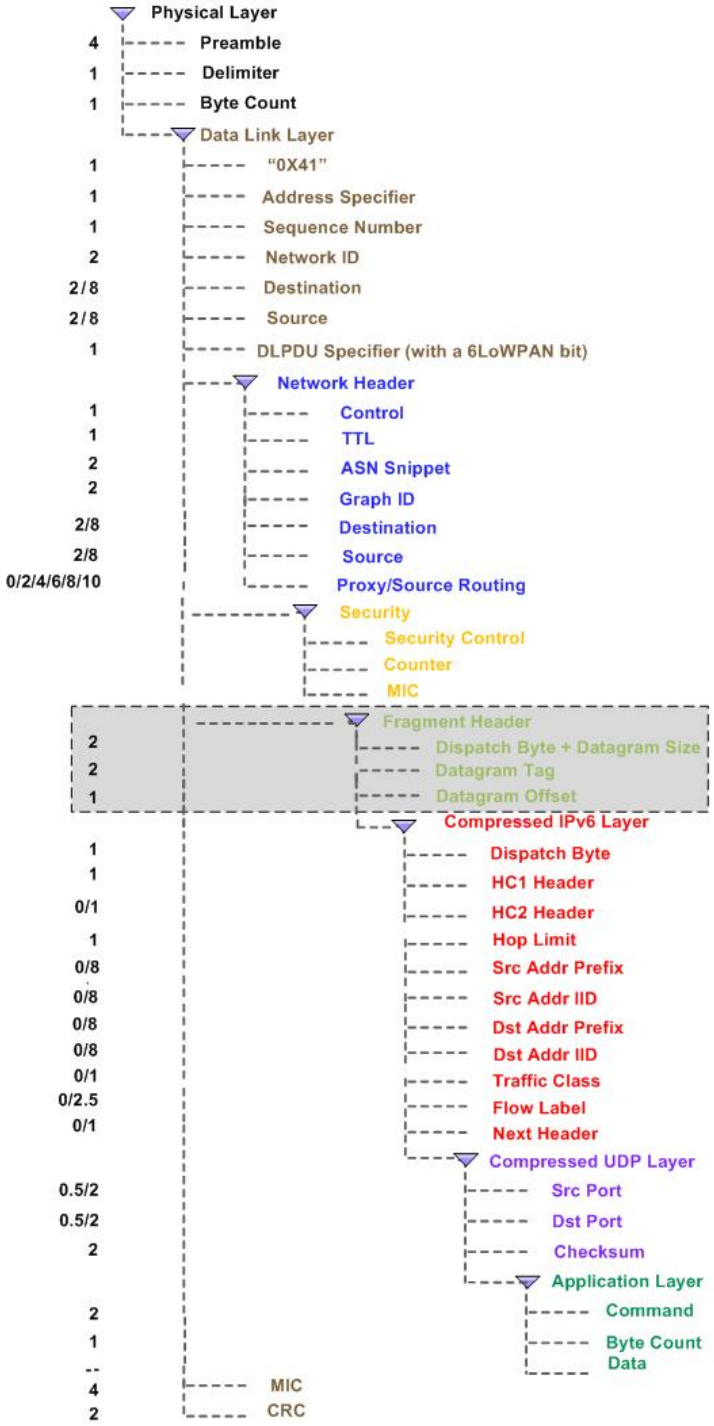

Figure 5.3: The format of IP-enabled WirelessHART packet

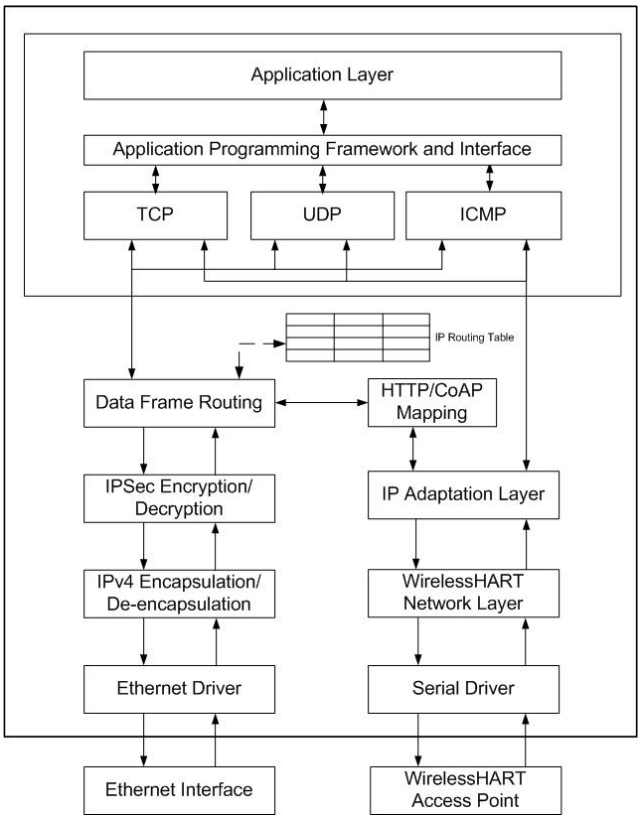

Figure 5.4: Design of the enhanced WirelessHART Gateway

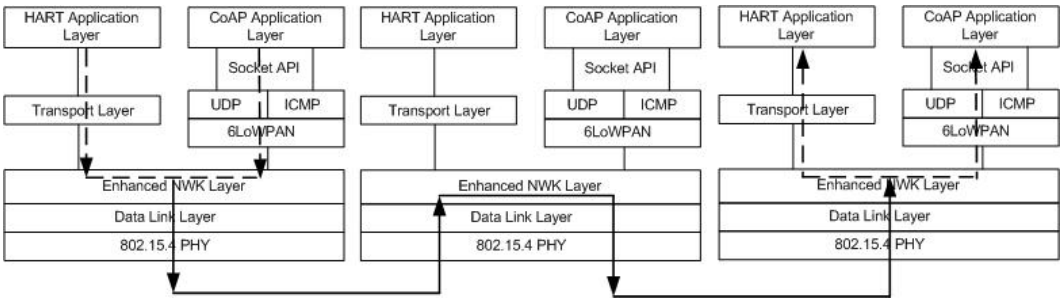

Figure 5.5: Mesh-under routing in IP-enabled WirelessHART networks

conveyed over bandwidth-limited wireless embedded networks to Internet Host by header compression/decompression and fragmentation/reassembly; the 6to4 tunneling module helps set up an IPv6-IPv4 tunnel between the Gateway and the Internet Host to communicate with; and the Host-to-Host IPSec module can help set up a secure communication between the Gateway and the to be communicated Internet Host.

5.1.3 OpenFlow Network

Achieving reliable and real-time inter-CPS services over physically distributed cyber-physical subsystems is not a trivial problem because the current Internet architecture has no facility to guarantee minimum bandwidth and end-to-end latency for network flows.

To address this problem, we propose to enhance the existing Internet architecture by deploying OpenFlow network to connect cyber-physical subsystems together. OpenFlow is an open standard that provides QoS support and enables researchers to run experimental protocols in campus networks. To deploy an OpenFlow network, it requires two key components, the inter-connected OpenFlow switches and the OpenFlow controller.

The hardware of OpenFlow switch is similar to current Ethernet switch, where we enhance the switch to support OpenFlow protocol. OpenFlow protocol provides an interface for remote controller to program the data plane of switches, which determine the corresponding action of network flows. These actions could be forwarding a packet to specific port, dropping out a packet, or putting it to a queue. By manipulating the performed actions on the switches, we can provide QoS guarantee to network flows. OpenFlow controller is usually a PC, where we can specify high level QoS requirement, then it can interpret it into low level actions and program these actions through OpenFlow protocol into OpenFlow switches.

5.1.4 Routing and End-to-End Secure Communication

Many existing wireless protocols support multi-hop mesh topology. This is achieved either through link-layer forwarding (Mesh-Under) or using IP routing (Route-Over). In our networking infrastructure, we apply the Mesh-Under approach. We keep the routing approach adopted in each cyber-physical subsystem unchanged and all fragments are delivered to the corresponding Gateway where reassembling and decompression happens. With this approach, intermediate devices on the route can be normal devices thus it completely supports incremental deployment. Figure 5.5 illustrates our mesh-under routing approach in IP-enabled WirelessHART networks.

On the other hand, end-to-end secure communication is crucial for many applications in cyber-physical systems. Since most security mechanisms designed for the Internet is too complicated for embedded devices, we only establish IPSec service between the edge routers and web servers in the Internet through tunneling mode. Inside each cyber-physical subsystem, the originally adopted security mechanism is used for encryption and authentication with no changes.

5.2 Testbed Implementation

To prove the concept and validate our design on IP-enhanced embedded device and Gateway, we built up an experimental testbed. As shown in Figure 5.6, the prototype system includes a CoAP-HTTP server, an enhanced WirelessHART Gateway, and a WirelessHART network consisting of three normal devices and four IP-enabled devices. In the experiments, we programmed all seven devices to publish WirelessHART messages to the Gateway periodically. In the meanwhile, all the IP-enabled devices are further configured to PUT their device information and sensor measurements through IP packets to the CoAP-HTTP server. The server supports secure communication over Internet through HTTPS. Through web browser, valid users can not only access the device information and their measurements in

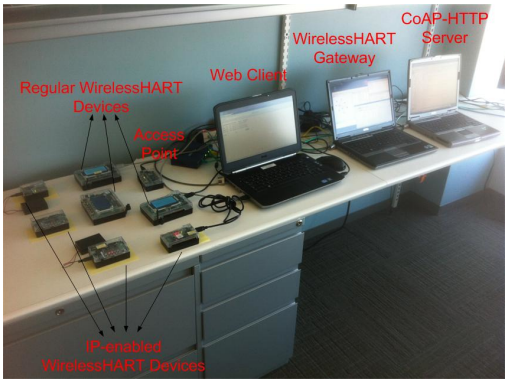

Figure 5.6: An overview of the IP-over-WirelessHART testbed

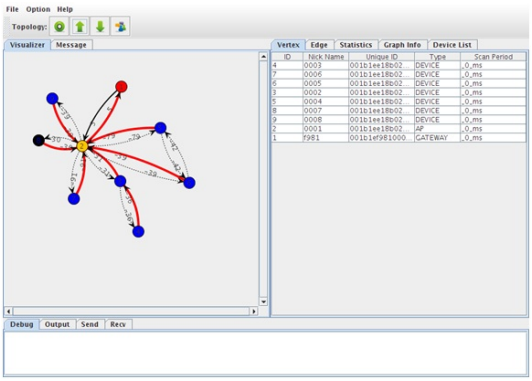

Figure 5.7: A network topology including both normal devices and IP-enabled devices

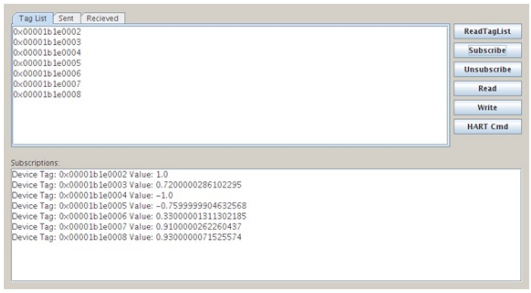

Figure 5.8: Both normal and IP-enabled devices support WirelessHART subscription

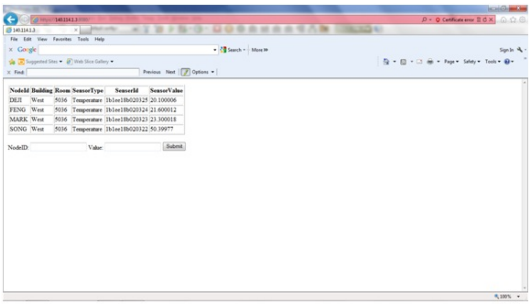

Figure 5.9: Users can monitor and control embedded devices through web browser

real-time manner by connecting to the server but also can configure the IP-enabled devices in an active way. Figure 5.7 shows that IP-enabled devices can join into the WirelessHART network in the exactly same way as normal devices and all the seven devices form a mesh topology. Figure 5.8 is a screen capture of a WirelessHART Host application which subscribes all the seven devices through standard WirelessHART commands. Figure 5.9 gives a screen capture of the user web browser which connects to the CoAP-HTTP server and display the real-time measurements of the IP-enabled embedded devices.

As the ongoing work, we are building a larger scale cyber-physical system which consists of multiple subsystems running different wireless communication protocols and connected through OpenFlow network. We are evaluating the system performance in main-

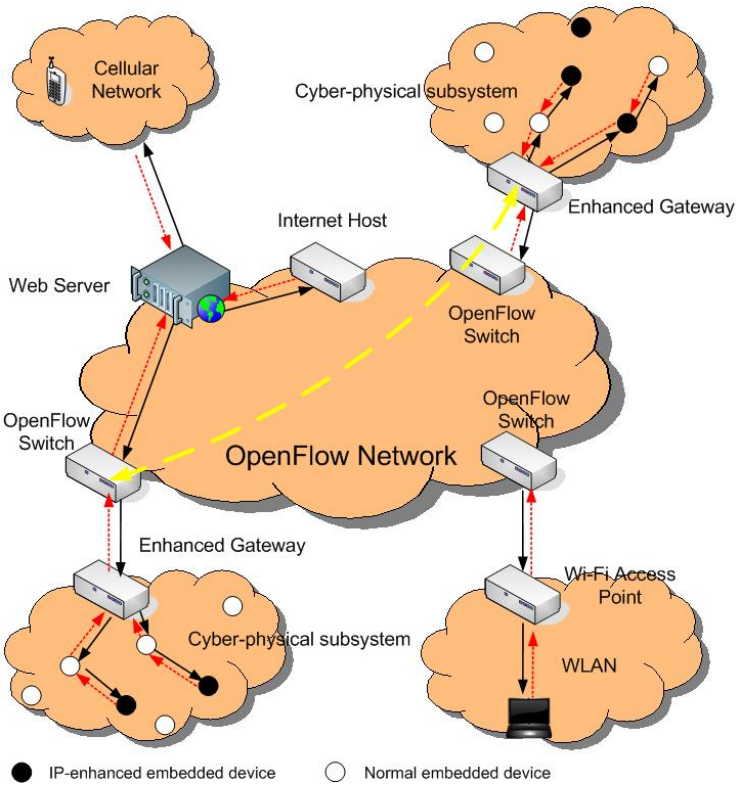

Figure 5.10: Testbed under deployment to achieve reliable and real-time inter-CPS services

taining reliable and real-time services for inter-CPS communication. Figure 5.10 shows the architecture of the system that we are building.

Chapter 6

Deferrable Scheduling Algorithms for Maintaining Data Freshness

Cyber-physical systems usually comprise a large network of distributed embedded sensors and actuators attached to physical entities for monitoring and control purposes. They are applied in many application domains that require timely processing of a massive amount of real-time data. Examples of real-time data include sensor measurements in industrial process control systems, positions of aircrafts in an air traffic control system, and temperature as well as air pressure in an engine control environment. Such real-time data are typically stored in a real-time database system (RTDBS). They are used to model the current status of entities in a system environment. However, real-time data are different from traditional data stored in databases in that they have time semantics indicating that sampled values are valid only for a certain time interval [68, 55, 82]. The concept of *temporal validity* is first introduced in [68] to define the correctness of real-time data. A real-time data object is *fresh* (or *temporally valid*) if its value truly reflects the current status of the corresponding entity in the system environment. Each real-time data object is associated with a *validity interval* as the lifespan of the current data value defined based on the dynamic properties of the data object. A new data value needs to be installed into the database before the validity

interval of its old value expires, i.e., the old one becomes temporally invalid. Otherwise, the RTDBS cannot detect and respond to environmental changes timely.

To maintain temporal validity, *sensor update transactions*, which capture the latest status of the entities in the system environment, are generated to refresh the values of the real-time data objects periodically [68, 38, 95]. A sensor update transaction has an infinite number of periodic jobs, which have fixed length period and relative deadline. The update problem for periodic update transactions consists of two parts [95]: *(1) the determination of the sampling periods and deadlines of update transactions; and (2) the scheduling of update transactions*. Two methods have been proposed in prior work for minimizing the update workload while maintaining the data freshness. As explained in [68, 38], a simple method to maintain temporal validity of real-time data objects is to use the *Half-Half (HH)* scheme in which the update period for a real-time data object is set to be half of the validity interval length of the object. To further reduce the update workload, the *More-Less* (*ML*) scheme is proposed [95]. *ML* extends the period of a transaction to be larger than its relative deadline, and the sum of the two equals to the validity interval of the corresponding data object. With this period and deadline assignment, as long as the update transaction set is schedulable under *ML*, it will incur lower update workload compared to *HH*, and its temporal validity will be maintained. The details of *HH* and *ML* will be reviewed in Section 6.1.2.

HH and *ML* can guarantee the real-time data freshness as long as the update transactions are schedulable under the assigned periods and deadlines. However, these approaches are pessimistic on the deadline and period assignment in the sense that they use periodic task model that has a fixed period and deadline for each task, and the deadline is equivalent to the worst-case response time. To address this problem, in this chapter, we present two *Deferrable Scheduling* algorithms with the objective to further reduce the update workload [92, 91, 35]. We study the problem of data freshness maintenance for firm real-time update transactions in a single processor RTDBS. The first algorithm, called *DS-FP*, is designed for fixed priority systems. Distinct from the past work of *HH* and *ML*, which has a

fixed period and relative deadline for each transaction, *DS-FP* adopts a *sporadic* task model. In contrast to *ML* in which a relative deadline is always equivalent to the worst-case response time of a transaction, *DS-FP* dynamically assigns relative deadlines to transaction jobs by deferring the sampling time of a transaction job as much as possible while still guaranteeing the temporal validity of real-time data to be updated. The deferral of a job's sampling time results in a relative deadline that is less than its worst-case response time, thus increases the separation of two consecutive jobs. This helps reducing processor workload produced by update transactions. We prove that *DS-FP* is better than *ML* in terms of schedulability and present a necessary and sufficient condition for feasibility of a set of update transactions under *DS-FP* based on pattern analysis. To help reduce its online computational complexity, we proposes two DEferrable Scheduling with Hyperperiod (*DESH*) algorithms that create hyperperiods from the *DS-FP* schedule. The transaction set can be scheduled by repeating the hyperperiod. Our experimental study of *DS-FP* demonstrates that it is an effective algorithm for reducing the workload of real-time update transactions. Our results also verify the accuracy of our theoretical estimation of average processor utilization under *DS-FP* and the effectiveness of the two proposed overhead reduction algorithms.

Since *DS-FP* is a fixed priority scheduling algorithm, it is inapplicable to systems which assume dynamic priorities. To overcome this shortcoming, a dynamic enhancement of *DS-FP* called Deferrable Scheduling with Least Actual Laxity First (*DS-LALF*) is proposed for assigning priorities to the update jobs based on their actual laxities in the run time. The actual laxity of a job is a measure of the spare time the job has before it misses its deadline by considering the time needed for higher priority jobs to be executed. With dynamic priority assignment for the update jobs using the least actual laxity, a lower update workload may result. We present a necessary and sufficient condition for the feasibility of *DS-LALF*, along with a pattern search algorithm to find the shortest and earliest pattern in the *DS-LALF* schedule. Our experimental results show that *DS-LALF* has a much lower

online computational overhead and incurs lower update workload compared with *DS-FP*. Its schedulability is close to *DS-FP* but is much better than the approaches under periodic task model, such as *HH* and *ML*.

The rest of the chapter is organized as follows: Section 6.1 reviews the background and existing approaches for maintaining freshness of real-time data. In Section 6.2, the Deferrable Scheduling algorithm for Fixed Priority transactions (*DS-FP*) is proposed and its overhead reduction algorithms and schedulability analysis are discussed in Section 6.3 and Section 6.4 respectively. Section 6.5 presents the Deferrable Scheduling algorithm with Least Actual Laxity First (*DS-LALF*) and its schedulability analysis. Finally, we conclude this chapter in Section 6.6.

6.1 Background and Related Work

Real-time data, whose state may become invalid with the passage of time, need to be refreshed by sensor update transactions generated by intelligent sensors that sample the value of real world entities. To monitor the states of entities faithfully, real-time data must be refreshed before they become invalid. The actual length of the temporal validity interval of a real-time data object is application dependent. For example, real-time data with validity interval requirements are discussed in [68, 69, 55]. One of the important design goals of RTDBSs is to guarantee that real-time data remain fresh, i.e., they are always valid.

6.1.1 Temporal Validity for Data Freshness

As real-time data values change continuously with time, the correctness of a real-time data object X_i depends on the difference between the real-time status $S(E_i)$ of the real world entity E_i and the current sampling value $Val(X_i)$ of X_i.

Definition 6.1.1: A real-time data object (X_i) at time t is temporally valid (or temporally consistent) if, for its j^{th} update finished last before t, the sampling time ($r_{i,j}$) plus the validity

interval ($\mathcal{V}_i$) of the data object is not less than t, i.e., $r_{i,j} + \mathcal{V}_i \geq t$ [68]. □

A data value for real-time data object X_i sampled at any time t will be temporally valid up to $(t + \mathcal{V}_i)$. Afterwards, it is invalid or called *stale*. The actual length of the *validity interval* of a real-time data object is application dependent and depends on the dynamic properties of the corresponding entity [68, 69, 55]. One of the important design goals of many real-time database systems is to guarantee that the real-time data remain fresh, i.e., they are always valid. Accessing stale data values could seriously affect the effectiveness of the real-time functions provided by the systems, e.g., generate incorrect responses even though the responses may be timely [70, 27, 43, 82].

6.1.2 Half-Half and More-Less

There have been extensive research works on maintaining the validity and freshness of real-time data [43, 21, 95, 49, 31, 89, 27, 18, 48, 14, 21]. Some of them use periodic update transactions while the others assume the arrival of update transactions to be sporadic. The second type of methods, e.g., [27, 18, 48, 14, 21], are mainly designed for soft real-time systems [70], and the main problem to be tackled is how to schedule update transactions at runtime to maximize the freshness of real-time data objects while minimizing their impact on the execution of real-time transactions from applications. Although these methods have been shown to be effective for achieving a better average performance, they cannot provide a guarantee on data validity for the execution of real-time transactions. On the contrary, the performance goal of those methods using periodic update transactions is to provide a guarantee on data validity. The main problems studied in these methods are: (1) how to determine the period and deadline for each update transaction to maintain validity of each real-time data object; and (2) how to define a schedule such that the deadlines of all the update transactions can be guaranteed. In this section, we will review two traditional approaches based on periodic task model for maintaining temporal validity, namely the *Half-Half* (*HH*) and *More-Less* (*ML*) approaches.

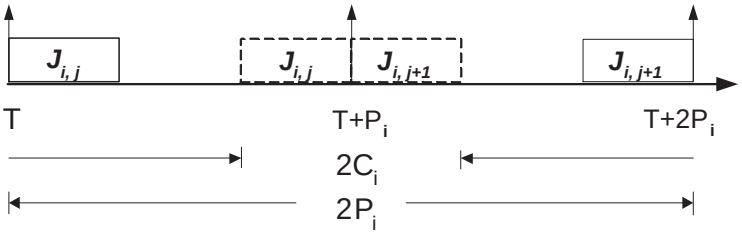

Figure 6.1: Extreme execution cases of jobs $J_{i,j}$ and $J_{i,j+1}$

In this chapter, $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ refers to a set of periodic update transactions $\{\tau_1, \tau_2, .., \tau_m\}$ and $\mathcal{X} = \{X_i\}_{i=1}^{m}$ refers to a set of real-time data objects. All real-time data objects are assumed to be kept in main memory. Associated with X_i $(1 \leq i \leq m)$ is a validity interval of length $\mathcal{V}_i$: transaction τ_i $(1 \leq i \leq m)$ updates the corresponding data object X_i.

Because each update transaction updates the different data object, no concurrency control is considered for update transactions. We assume that a sensor always samples the value of a real-time data object at the beginning of its period, and the system is *synchronous*, i.e., all the first jobs of update transactions are initiated at the same time. For convenience, let $d_{i,j}$, $f_{i,j}$ and $r_{i,j}$ denote the absolute deadline, completion time, and sampling (release) time of the j^{th} job $J_{i,j}$ of τ_i, respectively. We also assume that jitter between sampling time and release time of a job is zero (readers are referred to [95] for how jitters can be handled). Formal definitions of the frequently used symbols are given in Table 6.1. Deadlines of update transactions are firm deadlines. The goal of *Half-Half* and *More-Less*, which adopt *periodic* task model, is to determine period P_i and relative deadline D_i so that all the update transactions are schedulable and CPU workload resulting from periodic update transactions is minimized.

Both *HH* and *ML* assume a simple execution semantics for periodic transactions: a transaction must be executed once every period. However, there is no guarantee on when a job of a periodic transaction is actually executed within a period. Throughout this chapter we assume the scheduling algorithms are preemptive and ignore all preemption overhead. For convenience, we use terms transaction and task interchangeably in this chapter.

Symbol	Definition
X_i	Real-time data object i
τ_i	Update transaction updating X_i
$J_{i,j}$	The jth job of τ_i
$R_{i,j}$	Response time of $J_{i,j}$
C_i	Computation time of transaction τ_i
$\mathcal{V}_i$	Validity (interval) length of X_i
$f_{i,j}$	Finishing time of $J_{i,j}$
$r_{i,j}$	Release (Sampling) time of $J_{i,j}$
$d_{i,j}$	Absolute deadline of $J_{i,j}$
P_i	Period of transaction τ_i in *ML*
D_i	Relative deadline of transaction τ_i in *ML*
$P_{i,j}$	Separation of jobs (i.e., $r_{i,j+1} - r_{i,j}$) in *DS-FP*
$D_{i,j}$	Relative deadline of $J_{i,j}$ in *DS-FP*
$\overline{P}_i$	Average period of transaction τ_i in *DS-FP*
$\overline{D}_i$	Average relative deadline of transaction τ_i in *DS-FP*
$\overline{U}_{DS}$	Average processor utilization in *DS-FP*
$\Theta_i(a, b)$	Total cumulative processor demands from higher-priority transactions received by τ_i in interval $[a, b)$

Table 6.1: Symbols and definitions

Half-Half: In *HH*, the period and relative deadline of an update transaction are each typically set to be one-half of the data validity length [68, 38]. In Figure 6.1, the farthest distance of two consecutive jobs of τ_i (based on the sampling time $r_{i,j}$ of job $J_{i,j}$ and the deadline $d_{i,j+1}$ of its next job) is $2P_i$. If $2P_i \leq \mathcal{V}_i$, then the validity of real-time object X_i is guaranteed as long as jobs of τ_i meet their deadlines. Unfortunately, this approach incurs unnecessarily high CPU workload of the update transaction in the RTDBSs compared to *More-Less*.

More-Less: Consider the worst-case response time for any job of a periodic transaction τ_i where the response time is the difference between the transaction initiation time ($I_i + KP_i$) and the transaction completion time where I_i is the offset within the period.

Lemma 6.1.1: For a set of periodic transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ ($D_i \leq P_i$) with transaction initiation time ($I_i + KP_i$) ($K = 0, 1, 2, ...$), the worst-case response time for any job of τ_i

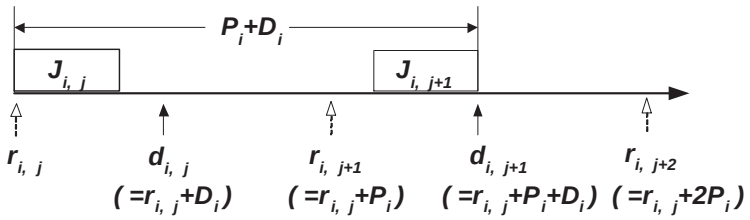

Figure 6.2: Illustration of *More-Less* scheme

occurs for the first job of τ_i when $I_1 = I_2 = ... = I_m = 0$. [52] □

For $I_i = 0$ $(1 \leq i \leq m)$, the transactions are *synchronous*. A time instant after which a transaction has the worst-case response time is called a *critical instant*, e.g., time 0 is a critical instant for all the transactions if those transactions are *synchronous*.

It should be noted that the first job response time is no longer the worst-case response time if $\exists i, (D_i > P_i)$ $(1 \leq i \leq m)$. This is illustrated by an example given by Lehoczky [51].

To minimize the update workload, *ML* is proposed to guarantee temporal validity [95], in which *Deadline Monotonic* (*DM*) [52] is used to schedule periodic update transactions. There are three constraints to follow for τ_i $(1 \leq i \leq m)$:

- *Validity constraint*: the sum of the period and relative deadline of transaction τ_i is always less than or equal to $\mathcal{V}_i$, i.e., $P_i + D_i \leq \mathcal{V}_i$, as shown in Figure 6.2.

- *Deadline constraint*: the period of an update transaction is assigned to be more than half of the validity length of the object to be updated, while its corresponding relative deadline is less than half of the validity length of the same object. For τ_i to be schedulable, D_i must be greater than or equal to C_i, the worst-case execution time of τ_i, i.e., $C_i \leq D_i \leq P_i$.

- *Feasibility constraint*: for a given set of update transactions, *Deadline Monotonic* scheduling algorithm [52] is used to schedule the transactions. Consequently, $\sum_{j=1}^{i}(\lceil \frac{D_i}{P_j} \rceil \cdot C_j) \leq D_i$ $(1 \leq i \leq m)$ if τ_i has higher priority than τ_j for $i < j$.

ML assigns priorities to transactions based on *Shortest Validity First* (SVF), i.e., in the *inverse* order of validity length and ties are resolved in favor of transactions with less slack (i.e., $\mathcal{V}_i - C_i$ for τ_i). It assigns deadlines and periods to τ_i as follows:

$$D_i = f_{i,0}^{ml} - r_{i,0}^{ml}, \tag{6.1}$$

$$P_i = \mathcal{V}_i - D_i, \tag{6.2}$$

where $f_{i,0}^{ml}$ and $r_{i,0}^{ml}$ are finishing and sampling times of the first job of τ_i, respectively. Note that in a synchronous system, $r_{i,0}^{ml} = 0$ and the first job's response time is the worst-case response time in *ML*. In this chapter, superscript *ML* is used to distinguish the finishing and sampling times in *ML* from those in *DS-FP*.

6.2 Deferrable Scheduling for Fixed Priority Systems (*DS-FP*)

A scheduler is *work-conserving* if it never idles the processor while there is a job awaiting execution. All schedulers discussed in this chapter are work-conserving. In this section, we present our *Deferrable Scheduling algorithm for Fixed Priority* transactions (*DS-FP*). Subsection 6.2.1 describes the thoughts that lead to the algorithm. Subsection 6.2.2 describes the *DS-FP* algorithm. Subsection 6.2.3 compares it with *ML*. Subsection 6.2.4 analyzes *DS-FP*'s utilization and Subsection 6.2.5 presents our experimental results to illustrate the effectiveness of *DS-FP*.

6.2.1 Intuition of *DS-FP*

In *ML*, D_i is determined by the first job's response time, which is the *worst-case* response time of all jobs of τ_i. Thus, *ML* is pessimistic on the deadline and period assignment in the sense that it uses periodic task model that has a fixed period and deadline for each task, and the deadline is equivalent to the worst-case response time. It should be noted

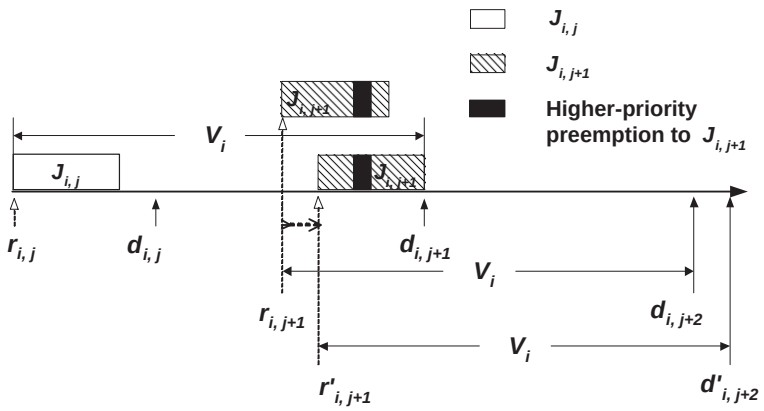

Figure 6.3: Illustration of *DS-FP* scheduling ($r_{i,j+1}$ is shifted to $r'_{i,j+1}$)

that the *validity constraint* can always be satisfied as long as $P_i + D_i \leq \mathcal{V}_i$. But processor workload is minimized only if $P_i + D_i = \mathcal{V}_i$. Otherwise, P_i can always be increased to reduce processor workload as long as $P_i + D_i < \mathcal{V}_i$. Given release time $r_{i,j}$ of job $J_{i,j}$ and deadline $d_{i,j+1}$ of job $J_{i,j+1}$ ($j \geq 0$),

$$d_{i,j+1} \leq r_{i,j} + \mathcal{V}_i \tag{6.3}$$

guarantees that the *validity constraint* can be satisfied, as depicted in Figure 6.3. Correspondingly, the following equation follows directly from Eq. 6.3.

$$(r_{i,j+1} - r_{i,j}) + (d_{i,j+1} - r_{i,j+1}) \leq \mathcal{V}_i. \tag{6.4}$$

If $r_{i,j+1}$ is shifted onward to $r'_{i,j+1}$ along the time line in Figure 6.3, it does not violate Eq. 6.4. This shift can be achieved, e.g., in the *ML* schedule, if preemption to $J_{i,j+1}$ from higher-priority transactions in $[r_{i,j+1}, d_{i,j+1}]$ is less than the worst-case preemption to the first job of τ_i. Thus, temporal validity can still be guaranteed as long as $J_{i,j+1}$ is completed by its deadline $d_{i,j+1}$.

The intuition of *DS-FP* is to defer the sampling time, $r_{i,j+1}$, of $J_{i,j}$'s subsequent job as late as possible while still guaranteeing the *validity constraint*. Note that the sampling

time of a job is also its release time, i.e., time that the job is ready to execute as we assume zero cost for sampling and no arrival jitter for a job for convenience of presentation.

The deferral of job $J_{i,j+1}$'s release time reduces the relative deadline of the job if its absolute deadline is fixed as in Eq. 6.3. For example, $r_{i,j+1}$ is deferred to $r'_{i,j+1}$ in Figure 6.3, but it still has to complete by its deadline $d_{i,j+1}$ in order to satisfy the *validity constraint* (Eq. 6.3). Thus its relative deadline, $D_{i,j+1}$, becomes $d_{i,j+1} - r'_{i,j+1}$, which is less than $d_{i,j+1} - r_{i,j+1}$. The deadline of $J_{i,j+1}$'s subsequent job, $J_{i,j+2}$, can be further deferred to $(r'_{i,j+1} + \mathcal{V}_i)$ to satisfy the *validity constraint*. Consequently, the processor utilization for completion of three jobs, $J_{i,j}$, $J_{i,j+1}$, and $J_{i,j+2}$ then becomes $\frac{3C_i}{2\mathcal{V}_i-(d_{i,j+1}-r'_{i,j+1})}$. It is less than the utilization $\frac{3C_i}{2\mathcal{V}_i-(d_{i,j+1}-r_{i,j+1})}$ required for completion of the same amount of work in *ML*.

Definition 6.2.1: Let $\Theta_i(a, b)$ denote the total cumulative processor demands made by all jobs of higher-priority transaction τ_j for $\forall j$ $(1 \leq j \leq i-1)$ during time interval $[a, b)$ from a schedule $\mathcal{S}$ produced by a fixed priority scheduling algorithm. Then,

$$\Theta_i(a, b) = \sum_{j=1}^{i-1} \theta_j(a, b),$$

where $\theta_j(a, b)$ is the total processor demands made by all jobs of single transaction τ_j during $[a, b)$. □

Next, we discuss how much a job's release time can be deferred. We shall use $r_{i,j+1}$ instead of $r'_{i,j+1}$ to denote the final deferred request time. According to fixed priority scheduling theory, $r_{i,j+1}$ can be derived backwards from its deadline $d_{i,j+1}$ as follows:

$$r_{i,j+1} = d_{i,j+1} - R_{i,j+1}(r_{i,j+1}, d_{i,j+1}); \tag{6.5}$$

$$R_{i,j+1}(r_{i,j+1}, d_{i,j+1}) = \Theta_i(r_{i,j+1}, d_{i,j+1}) + C_i. \tag{6.6}$$

Note that schedule of all higher-priority jobs that are released prior to $d_{i,j+1}$ needs to be

computed before $\Theta_i(r_{i,j+1}, d_{i,j+1})$ is computed. This computation can be invoked in a recursive process from jobs of lower-priority transactions to higher-priority transactions. Nevertheless, it does not require that schedule of all jobs should be constructed off-line before the task set is executed. Indeed, the computation of job deadlines and their corresponding release times is performed on-line while the transactions are being scheduled. We only need to compute the first jobs' response times when system starts. Upon completion of job $J_{i,j}$, deadline of its next job, $d_{i,j+1}$, is firstly derived from Eq. 6.3, then the corresponding release time $r_{i,j+1}$ is derived from Eq. 6.5. If $\Theta_i(r_{i,j+1}, d_{i,j+1})$ cannot be computed due to incomplete schedule information of release times and absolute deadlines from higher-priority transactions, *DS-FP* computes their complete schedule information on-line until it can gather enough information to derive $r_{i,j+1}$. Job $J_{i,j}$'s *DS-FP* scheduling information (e.g., release time, deadline, bookkeeping information, etc.) can be discarded after it completes and it is not needed by jobs of lower-priority transactions. This process is called *garbage collection* in *DS-FP*.

Let $S_J(t)$ denote the set of jobs of all transactions whose deadlines have been computed by time t. Also let $LSD_i(t)$ denote the latest scheduled deadline of τ_i at t, i.e., maximum of all $d_{i,j}$ for jobs $J_{i,j}$ of τ_i whose deadlines have been computed by t,

$$LSD_i(t) = \max_{J_{i,j} \in S_J(t)} \{d_{i,j}\} \ (j \geq 0). \tag{6.7}$$

Given job $J_{k,j}$ whose scheduling information has been computed at time t, and $\forall i \ (i > k)$, if

$$LSD_i(t) \geq d_{k,j}, \tag{6.8}$$

then the information of $J_{k,j}$ can be garbage collected.

Example 6.2.1: Suppose that there are three update transactions whose parameters are shown in Table 6.2. The resulting periods and deadlines in *HH* and *ML* are shown in the

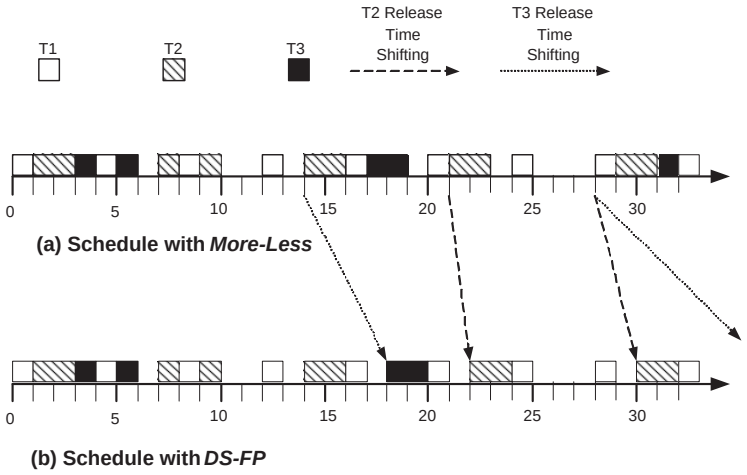

Figure 6.4: Comparing *ML* and *DS-FP* schedules

i	C_i	$\mathcal{V}_i$	$\frac{C_i}{\mathcal{V}_i}$	*ML*		*HH*
				P_i	D_i	$P_i(D_i)$
1	1	5	0.2	4	1	2.5
2	2	10	0.2	7	3	5
3	2	20	0.1	14	6	10

Table 6.2: Parameters and results for Example 6.2.1

same table. Utilizations of *HH* and *ML* are $\mathcal{U}_{ml} \approx 0.68$ and $\mathcal{U}_{hh} = 1.00$, respectively.

Figure 6.4 (a) and (b) depict the schedules produced by *ML* and *DS-FP*, respectively. It can be observed from both schedules that the release times of transaction jobs $J_{3,1}$, $J_{2,3}$, $J_{2,4}$, $J_{3,2}$ are shifted from times 14, 21, 28, 28 in *ML* to 18, 22, 30, 35 in *DS-FP*, respectively. □

The *DS-FP* algorithm is described in Section 6.2.2.

6.2.2 *DS-FP* Algorithm

This section presents the *DS-FP* algorithm. It is a *fixed priority* scheduling algorithm. Transaction priority assignment policy in *DS-FP* is the same as in *ML*, i.e., *Shortest Validity*

First. Given an update transaction set $\mathcal{T}$, it is assumed that τ_i has higher priority than τ_j if $i < j$ as we let $\mathcal{V}_i \leq \mathcal{V}_j$. Alg. 9 presents the *DS-FP* algorithm. For convenience of presentation, garbage collection is omitted in the algorithm. There are two cases for the *DS-FP* algorithm: 1) At system initialization time, Lines 13 to 22 iteratively calculate the first job's response time for τ_i. The first job's deadline is set as its response time (Line 23). 2) Upon completion of τ_i's job $J_{i,k}$ ($1 \leq i \leq m, k \geq 0$), the deadline of its next job ($J_{i,k+1}$), $d_{i,k+1}$, is derived at Line 29 so that the farthest distance of $J_{i,k}$'s sampling time and $J_{i,k+1}$'s finishing time is bounded by the validity length $\mathcal{V}_i$ (Eq. 6.3). Then the sampling time of $J_{i,k+1}$, $r_{i,k+1}$, is derived backwards from its deadline by accounting for the interference from higher-priority transactions (Line 31).

Function *ScheduleRT*($i,\ k,\ C_i,\ d_{i,k}$) (Alg. 10) calculates the release time $r_{i,k}$ with known computation time C_i and deadline $d_{i,k}$. It starts with release time $r_{i,k} = d_{i,k} - C_i$, then iteratively calculates $\Theta_i(r_{i,k},\ d_{i,k})$, the total cumulative processor demands made by all higher-priority jobs of $J_{i,k}$ during interval $[r_{i,k},\ d_{i,k})$, and adjusts $r_{i,k}$ by accounting for interference from higher-priority transactions (Lines 5 to 14). The computation of $r_{i,k}$ continues until interference from higher-priority transactions does not change in an iteration. In particular, Line 11 detects an infeasible schedule. A schedule becomes infeasible under *DS-FP* if $r_{i,k} < d_{i,k-1}$ ($k > 0$), i.e., release time of $J_{i,k}$ becomes earlier than the deadline of its preceding job $J_{i,k-1}$. Function *GetHPPreempt*($i,\ k,\ t_1,\ t_2$) scans interval $[t_1, t_2)$, adds up total preemptions from τ_j ($\forall j,\ 1 \leq j \leq i-1$), and returns $\Theta_i(t_1, t_2)$, the cumulative processor demands of τ_j during $[t_1, t_2)$ from schedule $\mathcal{S}$ that has been built.

Function *CalcHPPreempt*($i,\ k,\ t_1,\ t_2$) (Alg. 11) calculates $\Theta_i(t_1,\ t_2)$, the total cumulative processor demands made by all higher-priority jobs of $J_{i,k}$ during interval $[t_1,\ t_2)$. Line 7 indicates that ($\forall j, 1 \leq j < i$), τ_j's schedule is completely built before time t_2. This is because τ_i's schedule cannot be completely built before t_2 unless schedules of its higher-priority transactions are complete before t_2. In this case, the function simply returns amount of higher-priority preemptions for τ_i during $[t_1, t_2)$ by invoking *GetHPPreempt*($i,\ k,\ t_1,\ t_2$),

Alg 9 *DS-FP* algorithm

Input: A set of update transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ ($m \geq 1$) with known $\{C_i\}_{i=1}^{m}$ and $\{\mathcal{V}_i\}_{i=1}^{m}$.
Output: Construct a partial schedule $\mathcal{S}$ if $\mathcal{T}$ is feasible; Otherwise, reject.

1: **case** (system initialization time):
2: $t \leftarrow 0$; {// Initialization}
3: {// $LS\,D_i$ – Latest Scheduled Deadline of τ_i's jobs.}
4: $LS\,D_i \leftarrow 0,\ \forall i\ (1 \leq i \leq m)$;
5: $\ell_i \leftarrow 0,\ \forall i\ (1 \leq i \leq m)$;
6: {// ℓ_i is the latest scheduled job of τ_i}
7: **for** i=1 **to** m **do**
8: {// Schedule finish time for $\tau_{i,0}$.}
9: $r_{i,0} \leftarrow 0$;
10: $f_{i,0} \leftarrow C_i$;
11: {// Calculate higher-priority (HP) preemptions.}
12: oldHPPreempt $\leftarrow 0$; {// initial HP preemptions}
13: hpPreempt $\leftarrow$ **CalcHPPreempt**$(i, 0, 0, f_{i,0})$;
14: **while** (hpPreempt > oldHPPreempt) **do**
15: {// Accounting for the interference of HP tasks}
16: $f_{i,0} \leftarrow r_{i,0}$ + hpPreempt + C_i;
17: **if** $(f_{i,0} > \mathcal{V}_i - C_i)$ **then**
18: abort; {// The update transaction set is not schedulable.}
19: **end if**
20: oldHPPreempt $\leftarrow$ hpPreempt;
21: hpPreempt $\leftarrow$ **CalcHPPreempt**$(i, 0, 0, f_{i,0})$;
22: **end while**
23: $d_{i,0} \leftarrow f_{i,0}$;
24: **end for**
25: return;
26:
27: **case** (upon completion of $J_{i,k}$):
28: {// Schedule release time for $J_{i,k+1}$.}
29: $d_{i,k+1} \leftarrow r_{i,k} + \mathcal{V}_i$; {// get next deadline for $J_{i,k+1}$}
30: {// $r_{i,k+1}$ is also the sampling time for $J_{i,k+1}$}
31: $r_{i,k+1} \leftarrow$ **ScheduleRT**$(i, k+1, C_i, d_{i,k+1})$;
32: return;

Alg 10 ScheduleRT(i, k, C_i, $d_{i,k}$):

Input: $J_{i,k}$ with C_i and $d_{i,k}$.
Output: $r_{i,k}$.

```
1: oldHPPreempt ← 0; {// initial HP preemptions}
2: hpPreempt ← 0;
3: r_{i,k} ← d_{i,k} − C_i;
4: {// Calculate HP preemptions backwards from d_{i,k}.}
5: hpPreempt ← CalcHPPreempt(i, k, r_{i,k}, d_{i,k});
6: while (hpPreempt > oldHPPreempt) do
7:    {// Accounting for the interference of HP tasks}
8:    r_{i,k} ← d_{i,k} − hpPreempt − C_i;
9:    if (r_{i,k} < d_{i,k−1}) then
10:      abort; {// The update transaction set is not schedulable.}
11:   end if
12:   oldHPPreempt ← hpPreempt;
13:   hpPreempt ← GetHPPreempt(i, k, r_{i,k}, d_{i,k});
14: end while
15: return r_{i,k};
```

which returns $\Theta_i(t_1, t_2)$. If any higher-priority transaction τ_j ($j < i$) does not have a complete schedule during $[t_1,\ t_2)$, its schedule $\mathcal{S}$ up to or exceeding t_2 is built on the fly (Lines 14 to 19). This enables the computation of $\Theta_i(t_1,\ t_2)$. The latest scheduled deadline of τ_i's job, LSD_i, indicates the latest deadline of τ_i's jobs that have been computed.

The *worst-case* complexity of *ScheduleRT* is $O(m \cdot \mathcal{V}_m^2)$ assuming that $\frac{\mathcal{V}_m}{\mathcal{V}_1}$ is a constant. The following lemma states an important property of *ScheduleRT* when it terminates.

Lemma 6.2.1: Given a synchronous update transaction set $\mathcal{T}$ and *ScheduleRT*($i,\ k,\ C_i,\ d_{i,k}$) ($1 \le i \le m\ \&\ k \ge 0$), $LSD_l(t) \le LSD_j(t)$ ($k \ge l \ge j$) holds when *ScheduleRT*($i,\ k,\ C_i,\ d_{i,k}$) terminates at time t.

Proof: This can be proved by contradiction. Suppose that $LSD_l(t) > LSD_j(t)$ ($k \ge l \ge j$) when *ScheduleRT*($i,\ k,\ C_i,\ d_{i,k}$) terminates at t. Let $t_2 = LSD_l(t)$ at Line 14 in *CalcHPPreempt*. As $LSD_j(t) < t_2$ at the same line, *ScheduleRT* has not reached the point to terminate. This contradicts the assumption. □

Alg 11 CalcHPPreempt(i, k, t_1, t_2):

Input: $J_{i,k}$, and a time interval $[t_1, t_2)$.

Output: Total cumulative processor demands from higher-priority transactions τ_j $(1 \leq j \leq i-1)$ during $[t_1, t_2)$

1: $\ell_i \leftarrow k$; {// Record latest scheduled job of τ_i.}
2: $d_{i,k} \leftarrow t_2$;
3: $LSD_i \leftarrow t_2$;
4: **if** $(i = 1)$ **then**
5: {// No preemptions from higher-priority tasks.}
6: return 0;
7: **else if** $(LSD_{i-1} \geq t_2)$ **then**
8: {// Get preemptions from τ_j $(\forall j, 1 \leq j < i)$}
9: {// because τ_j's schedule is complete before t_2.}
10: return **GetHPPreempt**(i, k, t_1, t_2);
11: **end if**
12: {// build $\mathcal{S}$ up to or exceeding t_2 for τ_j $(1 \leq j < i)$.}
13: **for** j=1 **to** i-1 **do**
14: **while** $(d_{j,\ell_j} < t_2)$ **do**
15: $d_{j,\ell_j+1} \leftarrow r_{j,\ell_j} + \mathcal{V}_j$;
16: $r_{j,\ell_j+1} \leftarrow$ **ScheduleRT**$(j, \ell_j + 1, C_j, d_{j,\ell_j+1})$;
17: $\ell_j \leftarrow \ell_j + 1$;
18: $LSD_j \leftarrow d_{j,\ell_j}$;
19: **end while**
20: **end for**
21: return **GetHPPreempt**(i, k, t_1, t_2);

	τ_1	τ_2		τ_3	
Job	*ML* /*DS-FP*	*ML*	*DS-FP*	*ML*	*DS-FP*
0	(0,1)	(0, 3)	(0, 3)	(0, 6)	(0, 6)
1	(4,5)	(7, 10)	(7, 10)	(14, 20)	(18, 20)
2	(8,9)	(14, 17)	(14, 17)	(28, 34)	(35, 38)
3	(12,13)	(21, 24)	(22, 24)	...	...
4	(16,17)	(28, 31)	(30, 32)		
5	(20,21)	(35, 38)	(38, 40)		
6	(24,25)	...	...		
7	(28,29)				
8	(32,33)				
9	(36,37)				

Table 6.3: Release time and deadline comparison

The next example illustrates how *DS-FP* algorithm works with the transaction set in Example 6.2.1.

Example 6.2.2: Table 6.3 presents comparison of (release time, deadline) pairs of τ_1, τ_2 and τ_3 jobs before time 40 in Example 6.2.1, which are assigned by *ML* and *DS-FP* (Alg. 9). Note that τ_1 has same release times and deadlines for all jobs under *ML* and *DS-FP*. However, $J_{2,3}$, $J_{2,4}$, $J_{2,5}$, $J_{3,1}$, and $J_{3,2}$ have different release times and deadlines under *ML* and *DS-FP*. Alg. 9 starts at *system initialization* time. It calculates deadlines for $J_{1,0}$, $J_{2,0}$, $J_{3,0}$. Upon completion of $J_{3,0}$ at time 6, $d_{3,1}$ is set to $r_{3,0} + \mathcal{V}_3 = 20$. Then Alg. 9 invokes *ScheduleRT*$(3, 1, 2, 20)$ at Line 31, which will derive $r_{3,1}$. At this moment, Alg. 9 has already calculated the complete schedule up to $d_{3,0}$ (time 6). But the schedule in the interval $(6, 20]$ has only been partially derived. Specifically, only schedule information of $J_{1,0}$, $J_{1,1}$, $J_{1,2}$, $J_{1,3}$ $J_{2,0}$, and $J_{2,1}$ has been derived for τ_1 and τ_2. Alg. 10 (*ScheduleRT*) obtains $r_{3,1} = 20 - 2 = 18$ at Line 3, then invokes *CalcHPPreempt*$(3, 1, 18, 20)$. Alg. 11 (*CalcHPPreempt*) finds out that $LSD_2 = 10 < t_2 = 20$, then it jumps to the for loop starting at Line 13 to build the complete schedule of τ_1 and τ_2 in the interval $(6, 20]$, where the release times and deadlines for $J_{1,4}$, $J_{1,5}$, $J_{2,2}$, $J_{1,6}$, and $J_{2,3}$ are derived. Thus, higher-priority

transactions τ_1 and τ_2 have a complete schedule before time 20. Note that $r_{1,6}$ and $d_{1,6}$ for $J_{1,6}$ are derived when we calculate $r_{2,3}$ and $d_{2,3}$ such that the complete schedule up to $d_{2,3}$ has been built for transactions with priority higher than τ_2. As $r_{2,2}$ is set to 14 by earlier calculation, $d_{2,3}$ is set to 24. It derives $r_{2,3}$ backwards from $d_{2,3}$ and sets it to 22 because $\Theta_2(22, 24) = 0$. Similarly, $d_{3,1}$ and $r_{3,1}$ are set to 20 and 18, respectively. □

6.2.3 Comparison of *DS-FP* and *ML*

Note that *ML* is based on the *periodic* task model, while *DS-FP* adopts one similar to the *sporadic* task model [17]. However, the relative deadline of a transaction in *DS-FP* is not fixed. Theoretically, the separation of two consecutive jobs of τ_i in *DS-FP* satisfies the following condition:

$$\mathcal{V}_i - C_i \geq r_{i,j} - r_{i,j-1} \geq \mathcal{V}_i - WCRT_i \quad (j \geq 1), \tag{6.9}$$

where $WCRT_i$ is the worst-case response time of jobs of τ_i in *DS-FP*. Note that the maximal separation of $J_{i,j}$ and $J_{i,j-1}$ ($j \geq 1$), $\max_j\{r_{i,j} - r_{i,j-1}\}$, cannot exceed $\mathcal{V}_i - C_i$, which can be obtained when there are no higher-priority preemptions for the execution of job $J_{i,j}$ (e.g., the highest priority transaction τ_1 always has separation $\mathcal{V}_1 - C_1$ for $J_{1,j}$ and $J_{1,j-1}$). Thus, the processor utilization for *DS-FP* should be greater than $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$, which is the CPU workload resulting from the maximal separation $\mathcal{V}_i - C_i$ of each transaction.

if $f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$where $f_{i,0}^{ml}$ is the first job's response time (i.e., the worst-case response time) of τ_i's job in *ML*.

ML can be regarded as a special case of *DS-FP* in which sampling (or release) time $r_{i,j+1}^{ml}$ and deadline $d_{i,j+1}^{ml}$ ($j \geq 0$) can be specified as follows:

$$d_{i,j+1}^{ml} = r_{i,j}^{ml} + \mathcal{V}_i, \tag{6.10}$$

$$r_{i,j+1}^{ml} = d_{i,j+1}^{ml} - (\Theta_i(r_{i,0}^{ml}, f_{i,0}^{ml}) + C_i). \tag{6.11}$$

It is clear that $\Theta_i(r_{i,0}^{ml}, f_{i,0}^{ml}) + C_i = f_{i,0}^{ml}$ when $r_{i,0}^{ml} = 0$ $(1 \leq i \leq m)$ in *ML*.

Theorem 6.2.1: Given a synchronous update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ $(1 \leq i \leq m)$, if $(\forall i)\, f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$ in *ML*, then

$$WCRT_i \leq f_{i,0}^{ml}$$

where $WCRT_i$ and $f_{i,0}^{ml}$ denote the worst-case response time of τ_i under *DS-FP* and *ML*, respectively.

Proof: This can be proved by contradiction. Suppose that τ_k is the highest priority transaction such that $WCRT_k > f_{k,0}^{ml}$ holds in *DS-FP*. Also assume that the response time of $J_{k,n}$ $(n \geq 0)$, $R_{k,n}$, is the worst for τ_k in *DS-FP*. Note that schedules of τ_1 in *ML* and *DS-FP* are the same as in both cases, τ_1 jobs have the same relative deadline (C_1) and separation/period $(\mathcal{V}_1 - C_1)$. So $1 < k \leq m$ holds.

As $WCRT_k > f_{k,0}^{ml}$, there must be a transaction τ_l such that (a) τ_l has higher priority than τ_k $(1 \leq l < k)$; (b) at least two consecutive jobs of τ_l, $J_{l,j-1}$ and $J_{l,j}$, overlap with $J_{k,n}$, and (c) the separation of $J_{l,j-1}$ and $J_{l,j}$ satisfies the following condition:

$$r_{l,j} - r_{l,j-1} < \mathcal{V}_l - f_{l,0}^{ml}\ (j > 0), \tag{6.12}$$

where $\mathcal{V}_l - f_{l,0}^{ml}$ is the period (i.e., separation) of jobs of τ_l in *ML*.

Claim (a) is true because $k > 1$. It is straightforward that if each higher priority transaction of τ_k only has one job overlapping with $J_{k,n}$, then $R_{k,n} \leq f_{k,0}^{ml}$. This implies that Claim (b) is true. Finally, for $(\forall l < k)$ and $J_{l,j-1}$ and $J_{l,j}$ overlapping with $J_{k,n}$, if

$$r_{l,j} - r_{l,j-1} \geq \mathcal{V}_l - f_{l,0}^{ml}\ (j > 0),$$

then $R_{k,n} > f_{k,0}^{ml}$ cannot be true because the amount of preemptions from higher priority transactions received by $J_{k,n}$ in *DS-FP* is no more than that received by $J_{k,0}$ in *ML*. Thus,

Claim (c) is also true.

We know that release time $r_{l,j}$ in *DS-FP* is derived as follows:

$$r_{l,j} = d_{l,j} - R_{l,j} \tag{6.13}$$

where $R_{l,j}$ is the calculated response time of job $J_{l,j}$, i.e., $\Theta_l(r_{l,j}, d_{l,j}) + C_l$. Following Eq. 6.12 and 6.13,

$$\begin{aligned} d_{l,j} - R_{l,j} &= r_{l,j} \text{ {By Eq. 6.13}} \\ &< r_{l,j-1} + \mathcal{V}_l - f_{l,0}^{ml} \text{ {By Eq. 6.12}} \\ &= d_{l,j} - f_{l,0}^{ml} \text{ {By Eq. 6.3}} \end{aligned}$$

Finally,

$$R_{l,j} > f_{l,0}^{ml} \tag{6.14}$$

Eq. 6.14 contradicts the assumption that τ_k is the highest priority transaction that $WCRT_k > f_{k,0}^{ml}$ holds. Therefore, the theorem is proved. □

The following theorem gives a sufficient condition for schedulability of *DS-FP*.

Theorem 6.2.2: Given a synchronous update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ $(1 \le i \le m)$, if $(\forall i)\, f_{i,0}^{ml} \le \frac{\mathcal{V}_i}{2}$ in *ML*, then $\mathcal{T}$ is schedulable with *DS-FP*.

Proof: If $f_{i,0}^{ml} \le \frac{\mathcal{V}_i}{2}$, then the worst-case response times of τ_i $(1 \le i \le m)$ in *DS-FP*, $WCRT_i$, satisfy the following condition (by Theorem 6.2.1):

$$WCRT_i \le f_{i,0}^{ml} \le \frac{\mathcal{V}_i}{2}.$$

That is, $WCRT_i$ is no more than $\frac{\mathcal{V}_i}{2}$. Because the following three equations hold in *DS-FP*

according to Eq. 6.5 and 6.6:

$$r_{i,j} = d_{i,j} - R_{i,j}, \tag{6.15}$$

$$d_{i,j+1} = r_{i,j} + \mathcal{V}_i. \tag{6.16}$$

$$d_{i,j+1} = r_{i,j+1} + R_{i,j+1}, \tag{6.17}$$

Replacing $r_{i,j}$ and $d_{i,j+1}$ in Eq. 6.16 with Eq. 6.15 and 6.17, respectively, it follows that

$$r_{i,j+1} + R_{i,j+1} = d_{i,j} - R_{i,j} + \mathcal{V}_i.$$

That is,

$$r_{i,j+1} - d_{i,j} + R_{i,j+1} + R_{i,j} = \mathcal{V}_i. \tag{6.18}$$

Because

$$R_{i,j+1} + R_{i,j} \leq 2 \cdot WCRT_i \leq \mathcal{V}_i,$$

it follows from Eq. 6.18 that $r_{i,j+1} - d_{i,j} \geq 0$ holds. This ensures that it is schedulable to schedule two jobs of τ_i in one validity interval $\mathcal{V}_i$ under *DS-FP*. Thus $\mathcal{T}$ is schedulable with *DS-FP*. □

The following corollary states the correctness of *DS-FP*.

Corollary 6.2.1: Given a synchronous update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ ($1 \leq i \leq m$), if $(\forall i)\, f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$ in *ML*, then *DS-FP* correctly guarantees the *temporal validity* of real-time data.

Proof: As deadline assignment in *DS-FP* follows Eq. 6.3, the largest distance of two consecutive jobs, $d_{i,j+1} - r_{i,j}$ ($j \geq 0$), does not exceed $\mathcal{V}_i$. The *validity constraint* can be satisfied if all jobs meet their deadlines, which is guaranteed by Theorem 6.2.2. □

If $\mathcal{T}$ can be scheduled by *ML*, then by *ML* definition $(\forall i)\, f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$. Thus Corollary 6.2.2, which states a sufficient schedulability condition for *DS-FP*, directly follows from Theorem 6.2.2.

Corollary 6.2.2: Given a synchronous update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ ($1 \leq i \leq m$), if $\mathcal{T}$ can be scheduled by *ML*, then it can also be scheduled by *DS-FP*.

6.2.4 Theoretical Analysis of *DS-FP* Utilization

Next, we briefly review the approximation of average processor utilization of *DS-FP* from statistical perspective, provided that $\mathcal{T}$ can be scheduled by *ML*. Note that this implies that the approximation is applicable to transaction sets that all deadlines are no greater than their corresponding periods under *ML*. We then prove that such an approximation is a lower bound of CPU utilization for fixed priority schedules given the same priority assignment.

DS-FP does not schedule transactions periodically. Thus it is hard to derive its exact CPU utilization (unless the *DS-FP* schedule repeats). The following method provides an approximation of its average CPU utilization, which is quite close to the average CPU utilization obtained in the experiments in most cases. The CPU utilization approximation depends on the approximate values of the average deadline $\overline{D}$ and period $\overline{P}$ of transactions, which is described as follows.

Given a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$, let $\overline{U}_{DS}$ denote the average processor utilization under *DS-FP*, and $\overline{P}_j$ the average period for τ_j. The average relative deadline of τ_i, namely $\overline{D}_i$, is defined as follows:

$$\overline{D}_i = C_i + \sum_{j=1}^{i-1} [(\frac{\overline{D}_i}{\overline{P}_j}) \times C_j] \quad (1 \leq i \leq m). \tag{6.19}$$

Let $P_{i,j}$ and $D_{i,j+1}$ $(1 \leq i \leq m \ \wedge \ j \geq 0)$ denote $r_{i,j+1} - r_{i,j}$ and $d_{i,j+1} - r_{i,j+1}$ in Eq. 6.4, respectively. It follows that

$$P_{i,j} + D_{i,j+1} = \mathcal{V}_i. \tag{6.20}$$

Thus the following equation holds given arbitrarily large n ($n \to \infty$), where n is the number of jobs in averaging:

$$\overline{P}_i + \overline{D}_i = \mathcal{V}_i. \tag{6.21}$$

Following Eq. 6.19 and 6.21, $\overline{D}_i$ and $\overline{P}_i$ $(1 \leq i \leq m)$ can be calculated (from the highest priority transaction τ_1 to the lowest priority transaction τ_m) as following, respectively:

$$\overline{D}_i = \frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{\overline{P}_j}} \quad (1 \leq i \leq m) \tag{6.22}$$

$$\overline{P}_i = \mathcal{V}_i - \overline{D}_i \quad (1 \leq i \leq m) \tag{6.23}$$

Finally, $\overline{U}_{DS}$, the average utilization of the transaction set $\mathcal{T}$ under *DS-FP* can be defined as:

$$\overline{U}_{DS} = \sum_{i=1}^{m} \frac{C_i}{\overline{P}_i} = \sum_{i=1}^{m} \left(\frac{C_i}{\mathcal{V}_i - \frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{\overline{P}_j}}}\right) \tag{6.24}$$

Given transaction priorities, the following theorem states that $\overline{U}_{DS}$ is a CPU utilization lower bound for all fixed priority schedules satisfying the *validity constraint*.

Theorem 6.2.3: Given an update transaction set $\mathcal{T}$ and a priority assignment, $\overline{U}_{DS}$ is a processor utilization lower bound that can be achieved for any *fixed* priority schedules with the same priority assignment. In other words, given any fixed priority schedule S satisfying

the *validity constraint* for $\mathcal{T}$ with average processor utilization $\overline{U}_S$,

$$\overline{U}_{DS} \leq \overline{U}_S.$$

Proof: For convenience, let $\overline{U}^i_{DS}$ and $\overline{U}^i_S$ denote the average processor utilization of a set of transactions $\{\tau_1,..,\tau_i\}$ for *DS-FP* and S, respectively. Also let $\overline{D}^S_i$ and $\overline{P}^S_i$ denote average relative deadline and period of τ_i for S, respectively.

We prove it by induction starting from $\mathcal{T}_1 = \{\tau_1\}$, i.e., $m = 1$. Given $\tau_1 \in \mathcal{T}_1$ scheduled by *DS-FP*, it follows from Eq. 6.22 and 6.23 that

$$\overline{D}_1 = C_1,\ \overline{P}_1 = \mathcal{V}_1 - \overline{D}_1 = \mathcal{V}_1 - C_1.$$

It is clear that

$$\overline{U}^1_{DS} = \frac{C_1}{\overline{P}_1},$$

and $\overline{U}^1_{DS}$ is minimized because $\overline{P}_1$ is maximized when the *validity constraint* is satisfied. Thus, $\overline{U}^1_{DS} \leq \overline{U}^1_S$.

Given a set of transactions $\mathcal{T}_k = \{\tau_1,..,\tau_k\}$ (i.e., $m = k$), suppose that $\overline{U}^k_{DS}$ is minimized. We need to prove that $\overline{U}^{k+1}_{DS}$ is also minimized for a set of transactions $\mathcal{T}_{k+1} = \{\tau_1,..,\tau_k, \tau_{k+1}\}$ (i.e., $m = k + 1$). Since *DS-FP* is a fixed priority scheduling algorithm, the schedule of the k highest priority transactions $\tau_1,..,\tau_k$ in $\mathcal{T}_{k+1}$ is the same as the schedule of $\mathcal{T}_k$ under *DS-FP* due to the reason that the same priority assignment is applied to $\tau_1,..,\tau_k$ in both sets. This implies that utilizations of the set of transactions $\{\tau_1,..,\tau_k\}$ in $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$ are also the same under *DS-FP*. By Eq. 6.24, it follows that

$$\overline{U}^{k+1}_{DS} = \sum_{i=1}^{k+1} \frac{C_i}{\overline{P}_i} = \overline{U}^k_{DS} + \frac{C_{k+1}}{\overline{P}_{k+1}}. \tag{6.25}$$

By Eq. 6.22 and 6.23, it follows that

$$\overline{D}_{k+1} = \frac{C_{k+1}}{1 - \sum_{j=1}^{k} \frac{C_j}{\overline{P}_j}} = \frac{C_{k+1}}{1 - \overline{U}_{DS}^{k}}, \tag{6.26}$$

$$\overline{P}_{k+1} = \mathcal{V}_{k+1} - \overline{D}_{k+1}. \tag{6.27}$$

By assumption that $\overline{U}_{DS}^{k} \leq \overline{U}_{S}^{k}$ holds for the set of transactions $\{\tau_1,..,\tau_k\}$, and Eq. 6.26,

$$\overline{D}_{k+1} \leq \overline{D}_{k+1}^{S}. \tag{6.28}$$

Following the *validity constraint*, S satisfies the following condition for deadline $d_{i,j+1}^{S}$ of job $J_{i,j+1}$ and release time $r_{i,j}^{S}$ of job $J_{i,j}$ $(1 \leq i \leq m \ \& \ j \geq 0)$:

$$d_{i,j+1}^{S} - r_{i,j}^{S} \leq \mathcal{V}_i. \tag{6.29}$$

Correspondingly,

$$(r_{i,j+1}^{S} - r_{i,j}^{S}) + (d_{i,j+1}^{S} - r_{i,j+1}^{S}) \leq \mathcal{V}_i. \tag{6.30}$$

Similar to how Eq. 6.21 is derived, it follows from Eq. 6.30 that

$$\overline{P}_i^{S} + \overline{D}_i^{S} \leq \mathcal{V}_i \quad (1 \leq i \leq m). \tag{6.31}$$

By Eq. 6.31,

$$\begin{aligned} \overline{P}_{k+1}^{S} &\leq \mathcal{V}_{k+1} - \overline{D}_{k+1}^{S} \\ &\leq \mathcal{V}_{k+1} - \overline{D}_{k+1} \ \{\text{By Eq. 6.28}\} \\ &= \overline{P}_{k+1} \ \{\text{By Eq. 6.27}\} \end{aligned}$$

Thus, both $\overline{U}_{DS}^{k}$ and $\frac{C_{k+1}}{\overline{P}_{k+1}}$ in Eq. 6.25 are minimized. We have $\overline{U}_{DS}^{k+1} \leq \overline{U}_{S}^{k+1}$. Therefore, it is proved that $\overline{U}_{DS} \leq \overline{U}_{S}$ holds.

□

Statistically, we claim that $\overline{U}_{DS}$ is a lower utilization bound for fixed priority transactions with the same priority assignment although it may not be the *tightest* lower bound; and the actual utilization produced by *DS-FP* is close to this bound in our experiments. In other words, *DS-FP* is close to optimal.

The following example illustrates how the average utilization is estimated.

Example 6.2.3: Given the transaction set in Table 6.2, we calculate the average relative deadline and period of τ_i ($i = 1, 2, 3$) as follows:

$$\overline{D}_1 = C_1 = 1,\ \overline{P}_1 = \mathcal{V}_1 - \overline{D}_1 = 4,$$
$$\overline{D}_2 = \frac{C_2}{1 - \frac{C_1}{\overline{P}_1}} = 2.7,\ \overline{P}_2 = \mathcal{V}_2 - \overline{D}_2 = 7.3,$$
$$\overline{D}_3 = \frac{C_3}{1 - (\frac{C_1}{\overline{P}_1} + \frac{C_2}{\overline{P}_2})} = 4.2,\ \overline{P}_3 = \mathcal{V}_3 - \overline{D}_3 = 15.8.$$

The average processor utilization is $\overline{U}_{DS} = \sum_{i=1}^{m} \frac{C_i}{\overline{P}_i} = 0.65$. Given the transaction set in Table 6.2, it can be verified that the processor utilization for the first 200 time units is 63%, which is very close to our theoretical estimation and lower than the processor utilization from *ML* (68%). □

Exploring the Optimal Algorithm

We have proven that *DS-FP* is an improvement over *ML*. The question remains if there is a better algorithm than *DS-FP*. We have proven in Section 6.2.4 that *DS-FP* is close to optimal in terms of minimizing CPU workload resulting from sensor update transactions from the statistical perspective. Intuitively *DS-FP* should be very close to if not the best algorithm

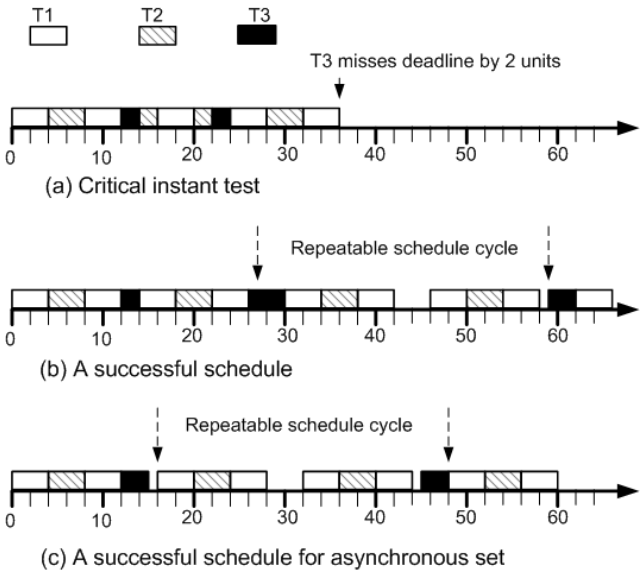

Figure 6.5: *DS-FP* not optimal

that approaches the utilization of $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$ because it always postpones the execution of a job as late as possible. Unfortunately, the answer may not be true because the answer to the following related statement is false: *For any transaction set, if it is schedulable by a fixed priority scheduler, then it could be scheduled by* DS-FP. We demonstrate this by a counter example.

Example 6.2.4: Let's consider a set of three transactions τ_1, τ_2 and τ_3. Their computation times are 4, 4, and 3 respectively; Their validity intervals are 12, 22, and 36 respectively. This set is not schedulable by *DS-FP* as it fails the critical instant test at time 0, shown in Figure 6.5(a). τ_3 misses deadline by 2 at time 36. However, if we schedule $J_{1,2}$ 2 time units earlier, we could successfully schedule the transaction set, shown in Figure 6.5(b). The key is that by doing so we postpone the start time of $J_{2,1}$ hence the deadline of $J_{2,2}$. The schedule between time 27 and 59 could be repeated forever. □

DS-FP fails the set in Example 6.2.4 because in the critical instant at time 0, we require that every transaction τ_i must finish the second job within $(0, \mathcal{V}_i)$. If we relax the

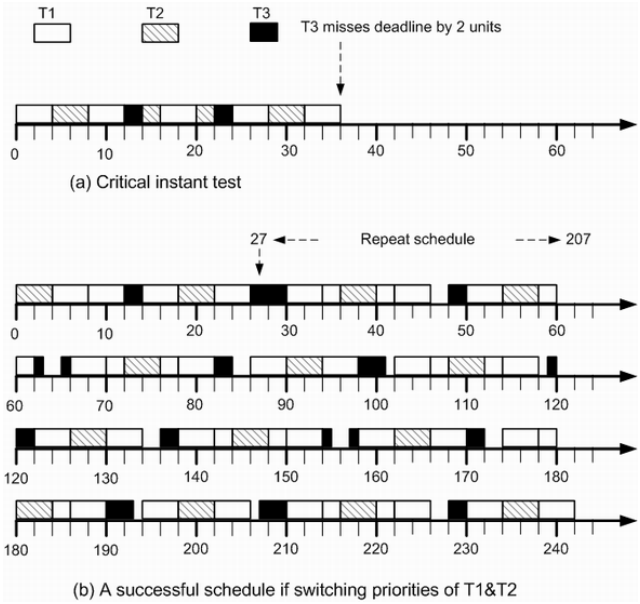

Figure 6.6: *DS-FP + RMA* not optimal

requirement and allow it to be finished within $(r_{i,0}, r_{i,0} + \mathcal{V}_i)$, *DS-FP* can schedule the set. This is shown in Figure 6.5(c), in which the schedule between time 16 and 48 is repeated. The question now becomes if *DS-FP* with this relaxation is still optimal.

Another interesting observation is that the set in Example 6.2.4 could be scheduled by *DS-FP* if we do not use *Shortest Validity First* priority assignment. If we switch the priorities of τ_1 and τ_2, *DS-FP* schedules the set as shown in Figure 6.6(b), in which the schedule between time 27 and 207 is repeated.

6.2.5 Performance Evaluation

This section presents important results from our experimental studies of the proposed deferrable scheduling algorithm (*DS-FP*). We first summarize our simulation model and parameters, and then compare *DS-FP* with the More-Less (*ML*) algorithm, which outperforms Half-Half [95]. Our goal is to find out how many transactions can be accommodated by each algorithm.

Simulation Model and Parameters

We have conducted two sets of experiments comparing the performance of *DS-FP* and *ML*. In the first set of experiments, we compare the update transaction workloads produced by *DS-FP* and *ML*. It is demonstrated that *DS-FP* produces lower CPU workload than *ML*. Also, we demonstrate that the increase of average sampling period from *DS-FP* is the main reason for its lower workload. In the second set of experiments, we study the performance of *DS-FP* and *ML* under mixed transaction workloads: a class of update transactions that maintain the freshness (validity) of real-time data objects, and a class of triggered transactions that are triggered by the changes of real-time data values. The triggered transactions need to read a group of real-time data objects for decision making. Given transactions belonging to different classes, update transactions are assigned higher priorities than the triggered transactions. For simplicity of the simulation study, only one version of a real-time data object is maintained. Upon refreshing a real-time data object, the older version is discarded. The primary performance metrics used in the experiments are CPU workload, mean transaction response time and missed deadline ratio (*MDR*) of the triggered transactions. We also measure the age of data (*AGE*) that indicates how old the real-time data object is at the commit time of a triggered transaction. Suppose that the current data value for real-time data object X_i is sampled at time t, and its data value is valid until $t + \mathcal{V}_i$. If a triggered transaction that reads the value of X_i commits at time t', its *AGE* is defined as:

$$AGE(X_i,\ t') = \begin{cases} \frac{t'-t}{\mathcal{V}_i} & : \quad t' > t + \mathcal{V}_i \\ 1 & : \quad t' \leq t + \mathcal{V}_i \end{cases}$$

A summary of the parameters and default settings used in experiments are presented in Tables 6.4 and 6.5. The baseline values for the parameters follow those used in [95], which are originally from air traffic control applications. Three classes of parameters are defined: system parameters, update transaction parameters and triggered transaction parameters. For system configurations, we only consider a single CPU, main memory based

Parameter Class	Parameters	Meaning
System	N_{CPU}	No. of CPU
	N_T	No. of real-time data objects
	$\mathcal{V}_i$	Validity interval of X_i
Update Transactions	C_i	CPU time for updating X_i
	Length	No. of data to update
Triggered Transactions	CPU Time	CPU time per data access
	Length	No. of data to access
	Arrival Rate	Transaction triggering rate
	Slack Factor	Transaction slack factor

Table 6.4: Experimental parameters

Parameter Class	Parameters	Meaning
System	N_{CPU}	1
	N_T	[50, 300]
	$\mathcal{V}_i(ms)$	[4000, 8000]
Update Transactions	$C_i(ms)$	[5, 15]
	Length	1
Triggered Transactions	CPU Time (ms)	[3, 5]
	Length	[5, 15]
	Arrival Rate	[5, 10]
	Slack Factor	8

Table 6.5: Experimental settings

RTDBS. The number of real-time data objects is varied from 50 to 300 and it is assumed that the validity interval length of each real-time data object is uniformly varied from 4000 to 8000 ms. For update transactions, it is assumed that each transaction updates one real-time data object, and the CPU time for each transaction is uniformly varied from 5 to 15 ms. For the triggered transactions, we assume that the number of real-time data objects accessed by each transaction is uniformly varied from 5 to 15, while accessing each data object takes 3 to 5 ms of CPU time. The inter-arrival time of the triggered transactions follows exponential distribution and the arrival rate is varied from 5 to 10 transactions per second. The

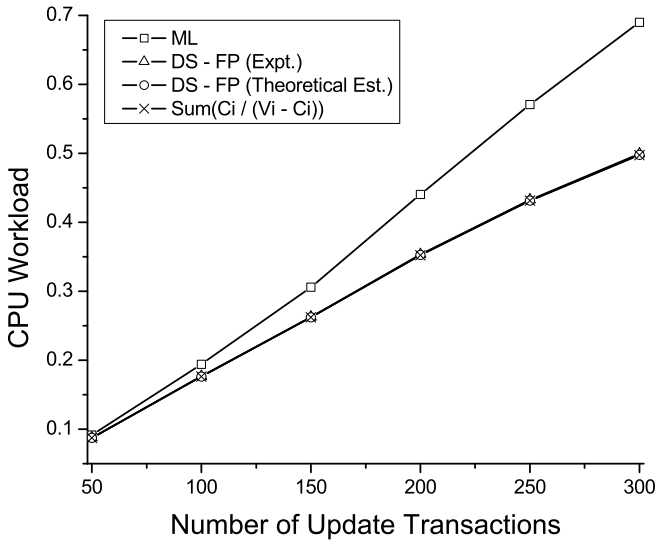

Figure 6.7: CPU workloads comparison

slack factor determines the slack of a transaction before its deadline expires and it is fixed at 8. Let $AT(\tau_i)$, $ET(\tau_i)$ and $Deadline(\tau_i)$ denote the arrival time, total execution time and deadline of triggered transaction τ_i, the deadline of τ_i can be calculated as follows in our experiments:

$$Deadline(\tau_i) = AT(\tau_i) + (ET(\tau_i) \times Slack\ Factor)$$

In the experiments, 95 percent confidence intervals have been obtained whose widths are less than ±5 percent of the point estimate for the performance metrics.

ML and *DS-FP*: Comparison of CPU Workloads

In this set of experiments, the CPU workloads of update transactions produced by *ML* and *DS-FP* are quantitatively compared. Update transactions are generated randomly according to the parameter settings in Table 6.5.

The resulting CPU workloads generated from *ML* and *DS-FP* are depicted in Figure 6.7. From the results, we observe that *DS-FP*'s CPU workload is consistently lower than that of *ML*. In fact, the difference widens as the number of update transactions increases. The difference reaches 18% when the number of transactions is 300. It is also observed that our experimental results of *DS-FP* match the average CPU workload calculated from the theoretical estimation (Eq. 6.24). Last but not least, the *DS-FP* CPU workload is only slightly higher than $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$, which is the CPU workload resulting from the maximal separation $\mathcal{V}_i - C_i$ $(1 \leq i \leq m)$ of each transaction. In fact, the difference is insignificant in Figure 6.7. The improvement of CPU workload under *DS-FP* is due to the fact that *DS-FP* adaptively samples real-time data objects at a lower rate. This is verified by the average sampling periods of update transactions obtained from experiments. Figure 6.8 shows the average sampling period for each transaction in *DS-FP* when the number of update transactions is 300. Given a set of update transactions, the period of transaction τ_i in *ML* (P_i^{ml}) is a constant and it can be calculated off-line [95], while the separation of sampling times of two consecutive jobs from the same transaction in *DS-FP* is dynamic and it is obtained on-line in the experiments. The mean value of the separations, i.e., the average sampling period, $\overline{P}_i^{ds}$, for transaction τ_i is calculated as follows, where n is the number of jobs generated by τ_i in the experiments.

$$\overline{P}_i^{ds} = \frac{1}{n-1} \sum_{j=1}^{n-1} (r_{i,j} - r_{i,j-1}) \tag{6.32}$$

In Figure 6.8, it is observed that $\overline{P}_i^{ds}$ is consistently larger than P_i^{ml} while the difference $(\overline{P}_i^{ds} - P_i^{ml})$ increases with the decrease of the transaction's priority. *DS-FP* reduces the average sampling rate more for lower-priority transactions, thus greatly reduces the workload of CPU. Figure 6.8 also demonstrates that the trend of $(\frac{\overline{P}_i^{ds}}{P_i^{ml}})$ increases similarly although it fluctuates.

In summary, when a set of update transactions is scheduled by *DS-FP* to maintain temporal validity of real-time data objects, it produces a schedule with a much lower

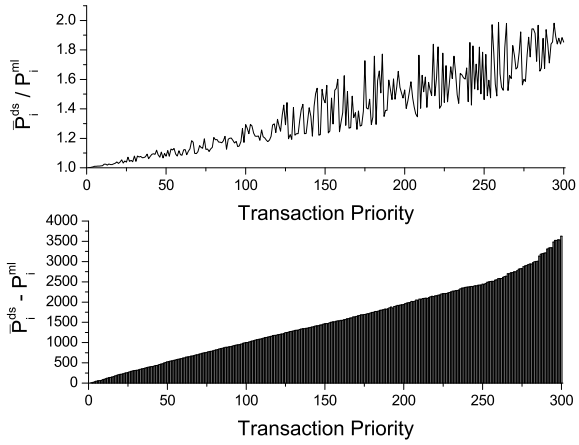

Figure 6.8: Sampling period comparison

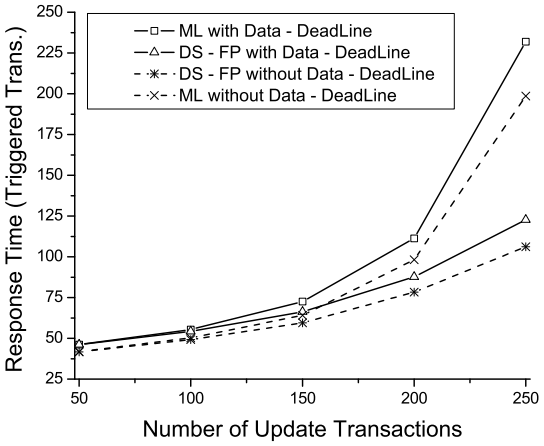

Figure 6.9: Response time comparison

CPU workload than *ML*. Thus more CPU capacity is available for other transactions, e.g., triggered transactions.

ML and *DS-FP*: Co-scheduling of Mixed Workloads

In this set of experiments, performances of mixed transactions are compared for *ML* and *DS-FP* in two scenarios: 1) Triggered transactions do not have deadlines. Their average response time and age of data at commit time are compared for *ML* and *DS-FP*; 2) Triggered transactions have deadlines. Their missed deadline ratios (MDRs) are compared for *ML* and *DS-FP*. In both cases, update transactions are scheduled by either *ML* or *DS-FP* for comparison. The parameter settings in the experiments are listed in Table 6.5.

Comparison of Average Response Time: In these experiments, triggered transactions do not have deadlines. We fix the triggered transactions' arrival rate at 10 transactions per second and the CPU time for accessing each real-time data object at 3 ms. We vary the number of update transactions from 50 to 250 in order to show its impact on the performance of average response times of the triggered transactions. We also consider two cases: 1) Triggered transactions obey *data-deadline* [96], which means that triggered transactions are

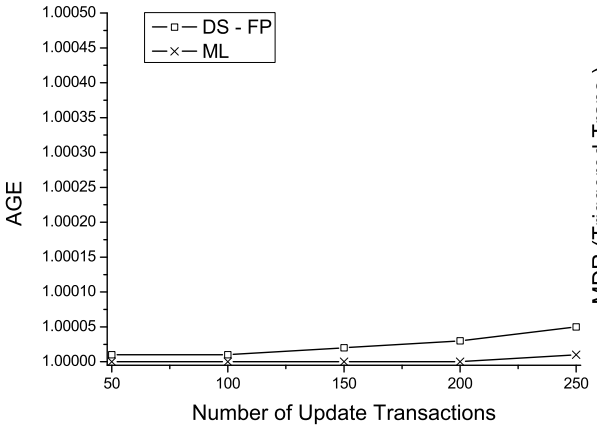

Figure 6.10: Average age of data

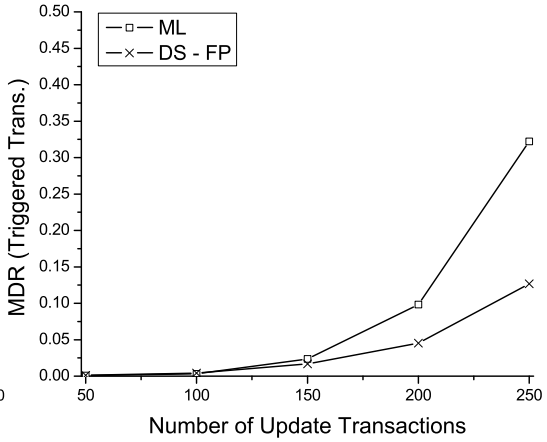

Figure 6.11: MDR comparison

aborted and restarted if the value of a real-time data object accessed by the transaction expires before the transaction commits; 2) Triggered transactions do not obey *data-deadline*, which means that the triggered transactions can still commit if the values of its accessed real-time data objects expire. Informally, data-deadline is a deadline assigned to a transaction due to the temporal constraints (i.e., validity interval length) of the data accessed by the transaction. For details of the concept of data-deadline, readers are referred to [96].

In Figure 6.9, the average response time of triggered transactions with update transactions scheduled by *DS-FP* is consistently lower than that of triggered transactions with update transactions scheduled by *ML*. For example, there is a 20% improvement in the response time of triggered transactions if the number of update transactions is 200 no matter whether *data-deadline* is obeyed or not. Figure 6.9 also demonstrates that the average response time of the triggered transactions in *ML* increases dramatically when the number of update transactions reaches 250. This is because the CPU is almost saturated by the workload generated from update transactions in the *ML* case if the number of update transactions reaches 250. However, the CPU workload of update transactions generated from *DS-FP* is much lower than that from *ML*.

Comparison of Average Age of Data: In these experiments, triggered transactions again do not have deadlines, and *data-deadline* is not obeyed by the triggered transactions. We compare the average age of data (AGE) accessed by the triggered transactions for *ML* and *DS-FP* at transaction commit time. Because *DS-FP* samples the data at lower rate, it is unclear how much impact the lower sampling rate has on the age of data at the commit time of a triggered transaction. Figure 6.10 demonstrates that *DS-FP*'s average age of data at commit time of the triggered transactions is only slightly larger than that of *ML*. In fact, the difference is very small and it can be totally ignored.

Comparison of Missed Deadline Ratio (MDR): Different from previous experiments, in this set of experiments we suppose that the triggered transactions have deadlines and they obey the *data-deadline* constraint. A triggered transaction that cannot commit before the validity of its accessed data expires has to be aborted, and restarted later if it has not missed its deadline. In such a case, a data-deadline is imposed on the triggered transaction due to the temporal constraints resulting from data validities. The triggered transactions are scheduled by the earliest deadline first (*EDF*) scheduling algorithm [54]. The CPU time for accessing a real-time data object is fixed at 5 ms. Figure 6.11 shows that the MDR of the triggered transactions under *DS-FP* is much lower than that under *ML*. For example, when the number of update transactions is 200 and the triggered transactions are scheduled by *EDF*, only 4% triggered transactions miss their deadlines under *DS-FP*, but around 10% triggered transactions miss their deadlines under *ML*. This is because the CPU workload of update transactions under *ML* is much higher than that under *DS-FP*. It can be observed that the difference of MDRs of triggered transactions widens as the number of update transactions increases.

In summary, *DS-FP* also provides better performance in the co-scheduling of mixed workloads where transactions can be triggered by the changes of values of real-time data objects. It greatly improves the average transaction response time and missed deadline ratio while only increases the average age of data insignificantly.

6.3 Overhead Reduction Algorithms for *DS-FP*

The *DS-FP* algorithm is very effective at reducing processor utilization while guaranteeing the validity constraint. However, its on-line scheduling overhead in terms of time complexity is much higher than that of *ML*. Moreover, the scheduling overhead of *DS-FP* depends on the number of transactions and the validity length of updated data. Thus, *DS-FP* can become very expensive, e.g., when the transaction set becomes large. In contrast to a periodic schedule that has the least common multiple of task periods as its hyperperiod, there is usually no such natural hyperperiod for a *DS-FP* schedule. In this section, we present two DEferrable Scheduling with Hyperperiod (*DESH*) algorithms for constructing periodic schedules off-line from the *DS-FP* algorithm so that the on-line scheduling time complexity can be reduced. Note that this will undoubtedly increase the space overhead for keeping the *DESH* schedules. But the space overhead can be kept reasonably low as demonstrated in our experiments in Section 6.3.3. Our *DESH* algorithms satisfy the following properties:

Property 1: A schedule satisfies the *validity constraint*. □

Property 2: The on-line scheduling time complexity is $O(1)$. □

6.3.1 DEferrable Scheduling with Hyperperiod: Schedule Construction

In this subsection, we present *DEferrable Scheduling with Hyperperiod* based on *Schedule Construction* (*DESH-SC*), a *DS-FP* based algorithm that can reduce the on-line scheduling overhead. The basic idea of *DESH-SC* is to search for an interval of *DS-FP* schedule, the hyperperiod, that could be repeated infinitely without violating the *validity constraint*. Note that *DESH-SC* could return without finding a hyperperiod.

The *DESH-SC* algorithm consists of two parts: an algorithm for finding the hyperperiod off-line (see Alg. 12) and an algorithm for scheduling transactions on-line. The latter is trivial once a hyperperiod is found because it only needs to repeat the hyperperiod schedule. Therefore, we next describe how the hyperperiod is derived from the schedule of *DS-FP*.

For a *DS-FP* schedule and a time period $[t_s, t_e]$, we say $[t_s, t_e]$ is a hyperperiod for the transaction set if for all transactions τ_i $(1 \leq i \leq m)$, the following schedule satisfies τ_i's *validity constraint*: it is the same as the *DS-FP* schedule from time 0 to t_e. From t_e onward, it repeats the *DS-FP* schedule in $[t_s, t_e]$ to infinity. Please note that t_s and t_e do not need to be idle time points in *DESH-SA*, which is different from the requirements of t_s and t_e in *DESH-SC*.

Lemma 6.3.1: $[t_s, t_e]$ is a hyperperiod in *DESH-SC* if for all τ_i $(1 \leq i \leq m)$ the following conditions hold.

1. t_s and t_e are CPU idle time points.
2. $t_s > \mathcal{V}_i$.
3. τ_i is scheduled at least once in $[t_s, t_e]$.
4. $I(t_s, \tau_i) \geq I(t_e, \tau_i)$, where function $I(t, \tau_i)$ is defined as the time distance between t and τ_i's latest release time before t.

Proof. Given any τ_i $(1 \leq i \leq m)$, we first prove in the hyperperiod schedule that the distance of the finish time of the first τ_i job after t_e and the release time of its latest job before t_e satisfies the *validity constraint*. Note that the first job after t_e repeats the first job in $[t_s, t_e]$. Because $t_s > \mathcal{V}_i$, the first job of τ_i in $[t_s, t_e]$ must finish by $(t_s - I(t_s, \tau_i)) + \mathcal{V}_i$, in other words, by $\mathcal{V}_i - I(t_s, \tau_i)$ after t_s. Since $[t_s, t_e]$ is repeated after t_e, the first job of τ_i after t_e also finishes by $\mathcal{V}_i - I(t_s, \tau_i)$ after t_e. The distance between the finish time of its first job in the *second* hyperperiod $[t_e, 2t_e - t_s]$ and the release time of its latest job in the *first* hyperperiod $[t_s, t_e]$ is *no longer* than:

$$I(t_e, \tau_i) + (\mathcal{V}_i - I(t_s, \tau_i)) = \mathcal{V}_i + (I(t_e, \tau_i) - I(t_s, \tau_i)) \leq \mathcal{V}_i.$$

Similarly, we can prove that the distance between the finish time of its first job in the $(k+1)^{th}$ $(k = 1, 2, ..)$ hyperperiod and the release time of its latest job in the k^{th} hyperperiod is *no longer* than $\mathcal{V}_i$. Thus, the jobs of τ_i satisfy the *validity constraint*. □

	τ_1	τ_2	τ_3
C_i	1	2	2
$\mathcal{V}_i$	5	10	20
P_i	4	7	14
D_i	1	3	6

Table 6.6: Parameters derived from *ML*

Note that the third condition could be derived from the fourth condition: If no job is scheduled in $[t_s, t_e]$, then τ_i's latest release time before t_s is also its latest release time before t_e, hence the fourth condition is always false.

To better satisfy the fourth condition, we need to assign t_s to be the end of an idle period and t_e to be the beginning of another idle period. By idle period $[t_1, t_2]$ we mean that CPU is busy right before t_1, it idles between t_1 and t_2, and it is busy again after t_2. Once the hyperperiod is found, we could increase t_e as long as the conditions in Lemma 6.3.1 are satisfied. The idea is presented in Alg. 12. In the algorithm, we continuously push t_2 of idle periods into a queue Q as possible candidates for t_s of a hyperperiod. For each subsequent idle period, we check its t_1 against each t_2 saved in Q to see if they form a hyperperiod. If the hyperperiod is found, we could then further increase t_e as long as $[t_s, t_e]$ still satisfies the conditions in Lemma 3.1. The increase cannot exceed $I(t_s, \tau_i) - I(t_e, \tau_i)$ for any transaction $\tau_i, 1 \leq i \leq m$. We define w to be the minimum of $I(t_s, \tau_i) - I(t_e, \tau_i)$ in the algorithm.
Note that Alg. 12 is not guaranteed to find a hyperperiod if one exists. This is due to the time limit ($t_{\max}$), space limit (the size of Q), and the value of $U_{\max}$. The value of $U_{\max}$ should not exceed 1.0. In our experiments, $U_{\max}$ is set as a value close to the average utilization estimation of *DS-FP* defined in Eq. 6.24 (Section 6.2.4). The following example demonstrates how the algorithm works.

Example 6.3.1: Suppose that there are three update transactions whose parameters are shown in Table 6.6. The resulting periods and deadlines in *ML* are shown in the same table. For each job of τ_1, τ_2 and τ_3 before time 45, Table 6.7 compares its (release time,

Alg 12 SearchHyperperiod

Input: A *DS-FP* schedule, a utilization limit $U_{\max}$, and a time limit $t_{\max}$.
Output: The hyperperiod with utilization $\leq U_{\max}$.

```
1:  U ← 1.001; {// Initialization of hyperperiod utilization.}
2:  t2 ← max{Vi | 1 ≤ i ≤ m};
3:  [t1, t2] ← first CPU idle period after t2;
4:  Append t2 to Q; {// Q is a FIFO queue of t2.}
5:
6:  while (U > Umax) do
7:     [t1, t2] ← next CPU idle period after t2;
8:     {// tmax is the maximum time to search.}
9:     if (t1 > tmax) then
10:       return failure;
11:    end if
12:    te ← t1;
13:    for ts = first in Q to last in Q do
14:       if ([ts, te] satisfies Condition 4 in Lemma 6.3.1) then
15:          Signal that a hyperperiod exists;
16:          w ← min{I(ts, τi) − I(te, τi) | 1 ≤ i ≤ m};
17:          te ← te + min(w, (t2 − t1)); {// Fine-tune te.}
18:          U' ← utilization in [ts, te];
19:          if (U' ≤ Umax) then
20:             GOTO line 29;
21:          end if
22:       end if
23:    end for
24:    if Q is full then
25:       Dequeue the oldest;
26:    end if
27:    Append t2 to Q;
28: end while
29: return [ts, te] as the hyperperiod;
```

	τ_1	τ_2		τ_3	
Job	*ML* /*DS-FP*	*ML*	*DS-FP*	*ML*	*DS-FP*
0	(0,1)	(0, 3)	(0, 3)	(0, 6)	(0, 6)
1	(4,5)	(7, 10)	(7, 10)	(14, 20)	(18, 20)
2	(8,9)	(14, 17)	(14, 17)	(28, 34)	(35, 38)
3	(12,13)	(21, 24)	(22, 24)	...	...
4	(16,17)	(28, 31)	(30, 32)		
5	(20,21)	(35, 38)	(38, 40)		
6	(24,25)	...	...		
7	(28,29)				
8	(32,33)				
9	(36,37)				

Table 6.7: Release time and deadline comparison

deadline) pairs which are assigned by *ML* and *DS-FP* respectively. Figure 6.12(a) depicts part of a *DS-FP* schedule during the interval $[15, 45]$, where t_1^i and t_2^i are the beginning and ending times of the $i^{th}(i = 1, 2, ...)$ idle period respectively. In the *DS-FP* schedule, the first idle period after $\mathcal{V}_3 = 20$, i.e., the largest $\mathcal{V}_i$, is $[21, 22]$. $t_2^1 = 22$ is pushed in Q. During the next three idle periods, the condition of Line 14 in Alg. 12 always fails. Note that τ_3 has not been scheduled yet since $\mathcal{V}_3$. For the fifth idle period $[t_1^5, t_2^5] = [41, 44]$, the second element in Q with $t_2^2 = 28$ satisfies the condition of Line 14 in Alg. 12. We have $t_s = 28$, $w = I(28, \tau_1) - I(41, \tau_1) = 3$, $t_e = 41 + \min(w, (t_2 - t_1)) = 41 + 3 = 44$, and $U' = 10/16 = 0.625$. If $U_{\max} \geq 0.625$, Alg. 12 returns hyperperiod $[28, 44]$. □

Note that Alg. 12 does not depend on which scheduling algorithm produces the input schedule. It could be applied to any valid schedule satisfying the *validity constraint*. The complete *DESH-SC* schedule constructed from Alg. 12 satisfies the Properties 1 and 2.

6.3.2 DEferrable Scheduling with Hyperperiod: Schedule Adjustment

In this subsection, we present *DEferrable Scheduling with Hyperperiod* based on *Schedule Adjustment* (*DESH-SA*), a *DS-FP* based algorithm that can reduce the on-line scheduling

overhead while achieving processor utilization close to that of *DS-FP*. By schedule adjustment, we mean changing release times and deadlines of jobs.

The basic idea of *DESH-SA* is to construct a hyperperiod schedule S_H off-line for $\mathcal{T}$, a set of validity constrained transactions. Suppose the first hyperperiod of the S_H schedule has length $\|S_H\|$. If the first hyperperiod of S_H can be constructed by adjusting the *DS-FP* schedule in the time interval $[0, \|S_H\|]$, the complete S_H schedule can be constructed by repeating the first hyperperiod of S_H infinitely every $\|S_H\|$ time units. Thus, similarly to *DESH-SC*, the *DESH-SA* algorithm consists of two parts: an algorithm for constructing the hyperperiod off-line (see Alg. 13) and an algorithm for scheduling transactions on-line. We next describe how the first hyperperiod schedule of S_H in the interval $[0, \|S_H\|]$ is derived from the schedule of *DS-FP*.

Given time $t_e > 0$, note that a *DS-FP* schedule in the interval $[0, t_e]$ can be constructed off-line. Assume that jobs J_{i,k_i-1} and J_{i,k_i} of τ_i ($k_i \geq 1$ & $1 \leq i \leq m$) satisfy the following condition for t_e:

$$d_{i,k_i-1} < t_e \leq d_{i,k_i}. \tag{6.33}$$

First, deadlines d_{i,k_i} will be adjusted to time t_e if they are different, i.e., all jobs J_{i,k_i} ($\forall i, 1 \leq i \leq m$) must complete by t_e. Note that there are $J_{i,0}, .., J_{i,k_i}$, i.e., $k_i + 1$ jobs of τ_i, during the interval $[0, t_e]$ after the adjustment, and k_i is the largest index of all jobs of τ_i that complete by t_e. Likewise, let jobs J_{i,k_i+1} ($\forall i, 1 \leq i \leq m$) of all τ_is after adjustment be released from time t_e, which is similar to the case that the first jobs are all released from time 0. Thus, jobs of all transactions can be released *synchronously* at time nt_e, where n is a natural number. This can be done by repeating schedule in $[0, t_e]$ forever.

Second, let the *DS-FP* schedule in $[0, t_e]$ be adjusted backwards from time t_e so that the adjusted schedule is valid for guaranteeing the *validity constraint*. The following condition can be enforced for $0 \leq t \leq t_e$ and integers $n_1, n_2 \geq 0$ if the schedule in the interval $[0, t_e]$ is repeated infinitely:

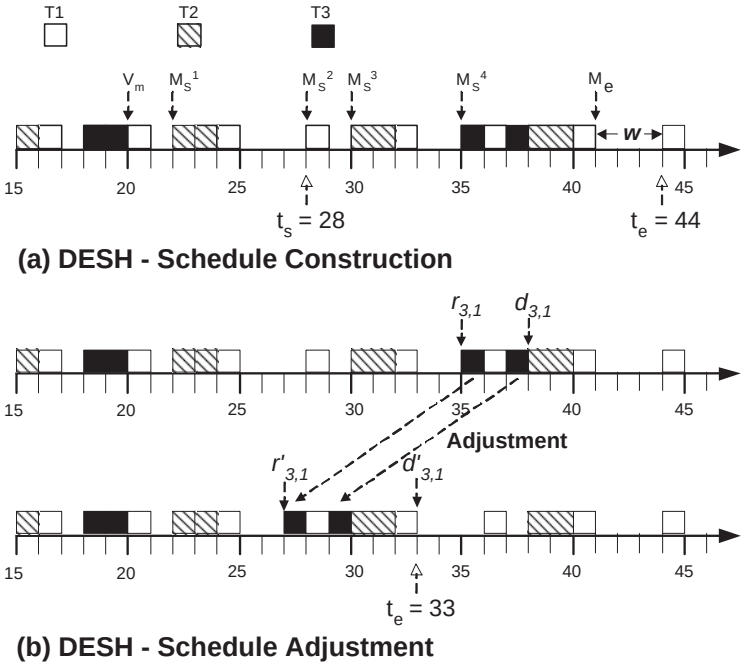

Figure 6.12: *DESH-SC* and *DESH-SA* examples

$$g(n_1 t_e, t) = g(n_2 t_e, t) \tag{6.34}$$

where $g(nt_e, t)$ is a function returning a pair of integers $< i, k >$, which indicates that the k^{th} job of τ_i in the n^{th} hyperperiod is active at time t (i.e., at time $nt_e + t$). Eq. 6.34 implies that any two hyperperiods have the exactly same schedule. Function $g(nt_e, t)$ is formally defined as

$$g(nt_e, t) = \begin{cases} < i, j - n(k_i + 1) >, & \text{the CPU is allocated to } J_{i,j} \text{ at time } nt_e + t; \\ < 0, 0 >, & \text{the CPU is idle at time } nt_e + t. \end{cases} \tag{6.35}$$

Note that n, j are integers, and $n \geq 0$ & $j \geq 0$ hold for Eq. 6.35. Eq. 6.34 ensures that all transactions are released *synchronously* at time 0, t_e, $2t_e$, ..., etc. If the processor is allocated to job $J_{i,j}$ at time $nt_e + t$, then it is the $(j - n(k_i + 1))^{th}$ job of τ_i from time nt_e (Note that there are $(k_i + 1)$ τ_i jobs during the interval $[0, t_e]$). Eq. 6.34 and 6.35 ensure that the complete S_H schedule is constructed periodically by repeating the schedule of the interval $[0, t_e]$ every t_e units.

Next, we present how time t_e is chosen according to the average processor utilization of *DS-FP*, $\overline{U}_{DS}$, as defined in Eq. 6.24.

First, t_e has to be a value that allows the processor utilization of the first S_H hyperperiod to be close to $\overline{U}_{DS}$, which is demonstrated to be very close to the average utilization of fixed priority transactions with the same priority assignment [92]. Any time $t \gg 0$ can be chosen as the initial value of t_e if the processor utilization at t_e, $U(t_e)$ is close to $\overline{U}_{DS}$, where $U(t_e)$ is defined as follows:

$$U(t_e) = \frac{\sum_{i=1}^{m}(k_i + 1)C_i}{t_e}. \tag{6.36}$$

Please note that k_i is the index of the last job (i.e., J_{k_i}) of transaction τ_i that completes by time t_e.

Second, t_e can be chosen from an idle time. The following theorem, which is a sufficient condition of constructing a general *DESH-SA* schedule for a validity constrained transaction set *without any adjustment*, explains the rationale for choosing an idle time as t_e.

Theorem 6.3.1: Given a *DS-FP* schedule for a validity constrained transaction set $\mathcal{T}$, suppose t_{idle} is an idle time in the schedule and the schedule before t_{idle} is feasible. Let r_{i,k_i-1} $(k_i \geq 1)$ be the latest release time of jobs of τ_i $(1 \leq i \leq m)$ before t_{idle}. If $\forall i$ $(1 \leq i \leq m)$,

$$t_{idle} - r_{i,k_i-1} + d_{i,0} \leq \mathcal{V}_i \tag{6.37}$$

holds, then the interval $[0, t_{idle}]$ can be used as the first hyperperiod of the *DESH-SA* schedule *without any adjustment*.

Proof. Note that once a job is released under *DS-FP*, the processor cannot be idle until the job completes. Thus,

$$\forall i (1 \leq i \leq m),\ d_{i,k_i-1} < t_{idle} < r_{i,k_i} \tag{6.38}$$

If all jobs J_{i,k_i} are released at time t_{idle}, i.e., $r_{i,k_i} = t_{idle}$, then the schedule of the interval $[t_{idle}, 2 \cdot t_{idle}]$ is the same as that of the interval $[0, t_{idle}]$. Moreover, if Eq. 6.37 holds,

$$\begin{aligned} d_{i,k_i} - r_{i,k_i-1} &= d_{i,k_i} - t_{idle} + t_{idle} - r_{i,k_i-1} \\ &= d_{i,0} - 0 + t_{idle} - r_{i,k_i-1} \\ &\leq \mathcal{V}_i \text{ \{By Eq. 6.37.\}} \end{aligned}$$

That is, two consecutive jobs J_{i,k_i-1}, J_{i,k_i} ($\forall i, 1 \leq i \leq m$) across two neighboring hyperperiods satisfy the validity constraint. Thus a feasible *DESH-SA* schedule can be constructed by having the schedule of the interval $[0, t_{idle}]$ as the first hyperperiod schedule. □

Note that if t_e is set to be t_{idle}, then it is not necessary to adjust the schedule of any transactions in the interval $[0, t_{idle}]$ for making the first hyperperiod of *DESH-SA*.

However, it is not always possible to find such a time t_{idle} for all transactions satisfying Eq. 6.37, in which case the *DS-FP* schedule in the interval $[0, t_e]$ corresponding to a subset of the transactions needs to be adjusted. Specifically, if transaction τ_h ($1 \leq h \leq m$) is the highest priority transaction whose schedule needs to be adjusted due to violation of Eq. 6.37, then the schedule of all lower-priority transactions τ_i ($h < i \leq m$) also needs to be adjusted due to the impact of release time and deadline adjustment of τ_i's higher-priority transactions in the interval $[0, t_e]$. This is described in Alg. 13.

Alg 13 AdjustScheduleForHyperperiod($\mathcal{T}$, t_e)

Input: Transaction set $\mathcal{T}$ and time $t_e > 0$.

Output: Adjusted schedule S_H in $[0, t_e]$ satisfying Eq. 6.34, and $\forall i, k_i$.

1: Construct *DS-FP* schedule S_H in $[0, t_e]$ for $\forall \tau_i \in \mathcal{T}$;
2: {// $J_{i,j}$ has $r_{i,j}, d_{i,j}$ computed in S_H ($j \leq k_i$ by Eq. 6.33).}
3: $h \leftarrow \min_i\{i \mid \tau_i \in \mathcal{T}$ & τ_i violates Eq. 6.37$\}$.
4: {// J_{i,k_i} is the latest τ_i job in the interval $[0, t_e]$.}
5: $\forall (i < h)$, $k_i \leftarrow k_i - 1$; {// No adjustment for $i < h$}
6: $t_s \leftarrow t_e$; {// Schedule in $[t_s, t_e]$ will be adjusted.}
7: **for** $i = h$ **to** m **do**
8: $\quad d'_{i,k_i} \leftarrow t_e$; {// $d'_{i,j}$ is adjusted from $d_{i,j}$.}
9: $\quad j \leftarrow k_i$;
10: $\quad$ **while** ($j > 0$) **do**
11: $\quad\quad$ **if** ($d'_{i,j} - r_{i,j} < \Theta_i(r_{i,j}, d'_{i,j}) + C_i$) {// $J_{i,j}$'s response time $> d'_{i,j} - r_{i,j}$.} **then**
12: $\quad\quad\quad r'_{i,j} \leftarrow d'_{i,j} - \Theta_i(r'_{i,j}, d'_{i,j}) - C_i$; {// $r'_{i,j}$ is adjusted from $r_{i,j}$.}
13: $\quad\quad\quad$ **if** ((($j < k_i$) **and** ($d'_{i,j+1} - r'_{i,j} > \mathcal{V}_i$)) **or** ($r'_{i,j} < 0$)) **then**
14: $\quad\quad\quad\quad$ report failure;
15: $\quad\quad\quad$ **end if**
16: $\quad\quad\quad$ **if** ($r'_{i,j} < d_{i,j-1}$) {// Ripple impact.} **then**
17: $\quad\quad\quad\quad d'_{i,j-1} \leftarrow r'_{i,j}$; {// Change $d_{i,j-1}$.}
18: $\quad\quad\quad$ **else**
19: $\quad\quad\quad\quad d'_{i,j-1} \leftarrow d_{i,j-1}$; {// No change.}
20: $\quad\quad\quad$ **end if**
21: $\quad\quad$ **else**
22: $\quad\quad\quad$ **if** ($t_s \geq d'_{i,j}$) {// No more adjustment for τ_i.} **then**
23: $\quad\quad\quad\quad t_s = d'_{i,j}$;
24: $\quad\quad\quad\quad$ break; {// Jump out of *while* loop}
25: $\quad\quad\quad$ **else** {// Examine the previous job of τ_i.}
26: $\quad\quad\quad\quad d'_{i,j-1} \leftarrow d_{i,j-1}$; {// No change.}
27: $\quad\quad\quad$ **end if**
28: $\quad\quad$ **end if**
29: $\quad\quad j \leftarrow j - 1$;
30: $\quad$ **end while**
31: $\quad$ **if** (($j == 0$) **and** (($d'_{i,j} < \Theta_i(0, d'_{i,j}) + C_i$))) **then**
32: $\quad\quad$ report failure;
33: $\quad$ **else**
34: $\quad\quad t_s \leftarrow 0$;
35: $\quad$ **end if**
36: **end for**
37: return adjusted S_H *in* $[0, t_e]$ and $\forall i, k_i$;

Note that Line 11 checks if $J_{i,j}$'s response time is greater than $d'_{i,j} - r_{i,j}$, the difference of its adjusted deadline and original release time. If so, its release time also needs to be adjusted backwards. This may cause a violation of the *validity constraint*, thus *DESH-SA* reports failure (Line 15). Otherwise, if its newly adjusted release time is ahead of its prior job's deadline, then its prior job's schedule has to be adjusted. This causes a ripple impact on the schedule adjustment. The next example illustrates *DESH-SA*.

Example 6.3.2: Figure 6.12(b) shows the result of applying *DESH-SA* on the same transaction set as in Example 6.3.1. The upper and lower schedules in Figure 6.12(b) are the *DS-FP* and its corresponding *DESH-SA* schedules, respectively. Let $t_e = 33$, then $h = 3$. Only τ_3 needs to be adjusted. *DESH-SA* sets $d'_{3,1} = 33$. $J_{3,1}$ can be re-scheduled and its adjusted release time is 27. The newly adjusted schedule from time 0 to 33 is the *DESH-SA* hyperperiod. □

After the schedule adjustment, the release time and deadline of job J_{i,k_i+1} ($\forall i, 1 \leq i \leq m$), which appears as the first job in the second hyperperiod, are set as follows:

$$r'_{i,k_i+1} = t_e; \tag{6.39}$$

$$d'_{i,k_i+1} = t_e + d_{i,0}. \tag{6.40}$$

Lemma 6.3.2: If $f^{ml}_{i,0} \leq \frac{\mathcal{V}_i}{2}$ in *ML*, the release time of J_{i,k_i} and the deadline of J_{i,k_i+1} after adjustment in Alg. 13 is constrained by $\mathcal{V}_i$, i.e.,

$$d'_{i,k_i+1} - r'_{i,k_i} \leq \mathcal{V}_i. \tag{6.41}$$

Proof. $\forall i (i < h)$, $(k_i - 1)$ is assigned to k_i at Line 5 in Alg. 13. By Theorem 6.3.1,

$$t_e + d_{i,0} - r_{i,k_i} \leq \mathcal{V}_i$$

$$\Rightarrow d'_{i,k_i+1} - r'_{i,k_i} \leq \mathcal{V}_i \{ \textit{By Alg. 13, } r'_{i,k_i} = r_{i,k_i}. \}$$

Next, we prove it for $h \leq i \leq m$. By schedule adjustment,

$$r'_{i,k_i+1} = d'_{i,k_i} = t_e. \tag{6.42}$$

After the schedule adjustment, according to Theorem 6.2.1, the following equations hold:

$$d'_{i,k_i+1} - r'_{i,k_i+1} \leq f^{ml}_{i,0}; \tag{6.43}$$

$$d'_{i,k_i} - r'_{i,k_i} \leq f^{ml}_{i,0}. \tag{6.44}$$

The sum of Eq. 6.43 and Eq. 6.44 is:

$$d'_{i,k_i+1} - r'_{i,k_i} \leq 2f^{ml}_{i,0} \leq \mathcal{V}_i.$$

Thus, Eq. 6.41 holds. □

Lemma 6.3.2 guarantees that the *validity constraint* can be satisfied for J_{i,k_i} and J_{i,k_i+1}, which are two consecutive jobs in *different* hyperperiods (e.g., the 1st and 2nd hyperperiods). Alg. 13 also guarantees that the *validity constraint* can be satisfied for jobs $J_{i,0}$, .., J_{i,k_i}, which are in the same hyperperiod. Moreover, if the schedule of the first S_H hyperperiod can be constructed off-line, the on-line scheduling overhead of *DESH-SA* in terms of time complexity is constant as it only needs to repeat the schedule of the first S_H hyperperiod infinitely. Therefore, the complete *DESH-SA* schedule constructed from Alg. 13 satisfies the aforementioned Properties 1 and 2.

Parameter	Meaning	Value
N_{CPU}	No. of CPU	1
N_T	No. of real-time data	[10, 300]
$\mathcal{V}_i$ (ms)	Validity interval of X_i	[4000, 8000]
C_i (ms)	Time for updating X_i	[5, 15]
Trans. length	No. of data to update	1

Table 6.8: Parameters and default settings

6.3.3 Performance Evaluation

This section presents important results from our simulation studies of the *DESH* algorithms. Our goal is to find out whether the *DESH* algorithms are effective for reducing the *DS-FP* overhead. The primary performance metrics used in our experimental studies are the CPU workload and the number of transactions supported in the system.

Simulation Model and Parameters

In the experiments, we investigate whether *DESH-SA* and *DESH-SC* can find a hyperperiod, and if so, how much excess CPU workloads they may incur compared to *DS-FP*. We also compare the hyperperiod length of *DESH-SA* and *DESH-SC* to study the space efficiency of the approaches, and demonstrate the percentage of transactions to be adjusted when we calculate the hyperperiod for *DESH-SA*.

For simplicity, only one version of a real-time data object is maintained. Upon refreshing a real-time data object, the older version is discarded. We ignore the on-line scheduling overhead in our experiments, and consider it to be $O(1)$ for all algorithms (which is true for *DESH* algorithms). This is in favor of *DS-FP* as its scheduling overhead is ignored for the CPU workload in our experiments. We define N_{adjust} to be the average number of jobs whose release times or deadlines are adjusted in $[0, t_e]$ under *DESH-SA*.

A summary of the parameters and default settings is presented in Table 6.8. Two categories of parameters are defined: system and update transaction parameters. For system

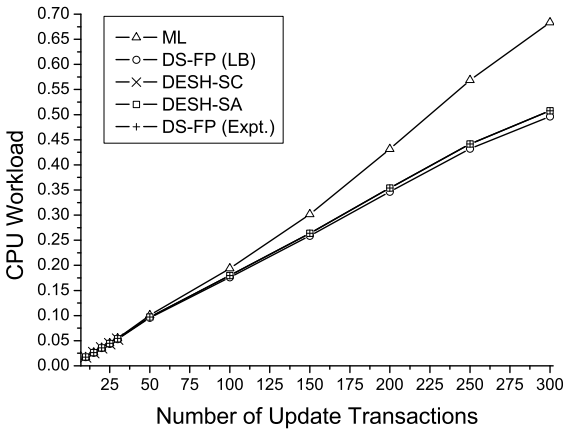

Figure 6.13: CPU workload comparison

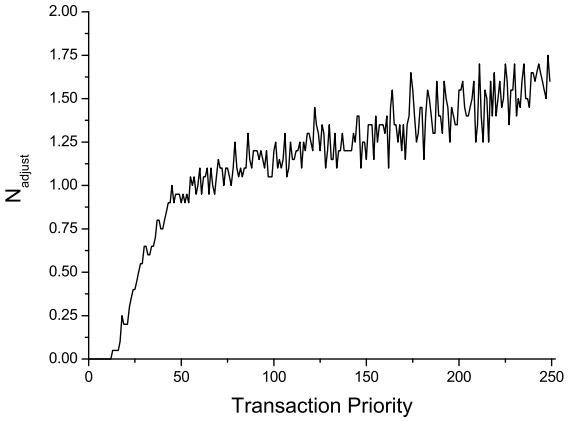

Figure 6.14: Average number of adjusted jobs

configurations, we only consider a single CPU, main memory based RTDBS. The number of real-time data varies from 10 to 300 and the validity length of each real-time data object is uniformly distributed from 4000 to 8000 ms. For update transactions, it is assumed that each transaction updates one real-time data object, and its CPU time is uniformly distributed from 5 to 15 ms.

Experimental Results

In this subsection, the CPU workloads of sensor update transactions produced by *DS-FP*, *DESH-SC* and *DESH-SA* are quantitatively compared. In the first set of experiments, update transactions are generated randomly according to the parameter settings in Table 6.8 while in the second set of experiments, $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i}$ of the update transaction set is fixed at 45% and $\frac{\mathcal{V}_i}{C_i}$ is varied to show its impact on the performance of the algorithms. With the *DESH* algorithms, the transaction with maximum validity interval is run at least 200 times (thus issuing 200 jobs) so that the CPU workload of the first $\mathcal{S}_{\mathcal{H}}$ hyperperiod is close to $\overline{U}_{DS}$. Then the idle times following the completion of those jobs under *DS-FP* are chosen as t_e candidates. Those candidates are tested to find out whether the interval $[0, t_e]$ can be used

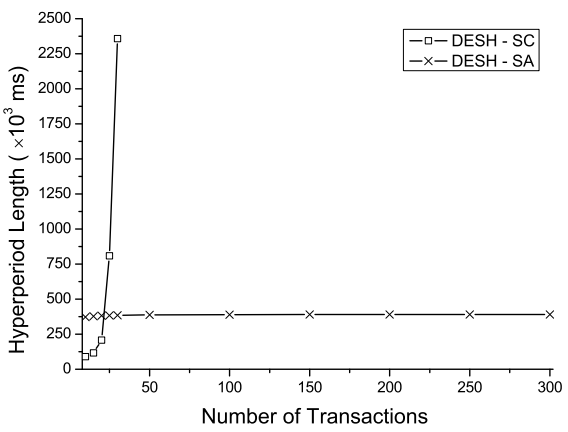

Figure 6.15: Average hyperperiod length

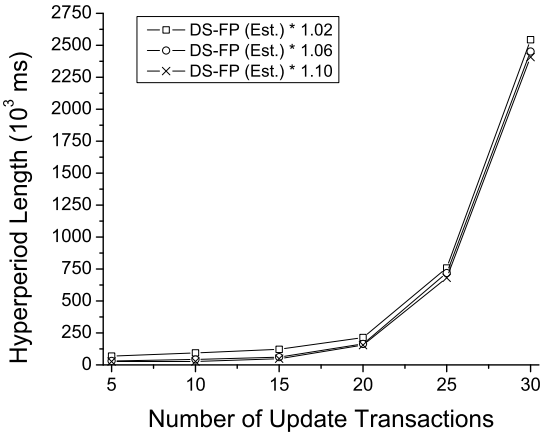

Figure 6.16: Hyperperiod length (*DESH-SC*)

as the first hyperperiod.

The resulting CPU workloads generated from *DESH-SA*, *DESH-SC* and *DS-FP* are depicted in Figure 6.13. In the figure, *DS-FP*(Est.) indicates the estimated average CPU utilization $\overline{U}_{DS}$, and *DS-FP*(Expt.) is the actual utilization of *DS-FP* obtained in the experiments. The results in Figure 6.13 demonstrate that the CPU workload of *DS-FP*(Est.) nearly matches that of *DS-FP*(Expt.). In Figure 6.13, the *DESH-SC* curve is only plotted in the range where the number of update transactions in the system varies from 5 to 35. This is because in our experiments, *DESH-SC* can only find its hyperperiod when the number of update transactions is no more than 35. Indeed, CPU workloads of *DESH-SC* and *DESH-SA* nearly match that of *DS-FP*, and it is hard to tell the difference in the figure. However, it is observed that the *DESH-SA* workload is consistently lower than that of *ML*. This is expected because the deadline adjustment in *DESH-SA* has little impact on increasing the overall system workload.

Even when the number of transactions in the system is up to 300, *DESH-SA* only incurs less than 1% excess CPU workload compared to that of *DS-FP*. Note that the online scheduling overhead is ignored for *DS-FP* in our experiments. Thus the actual CPU

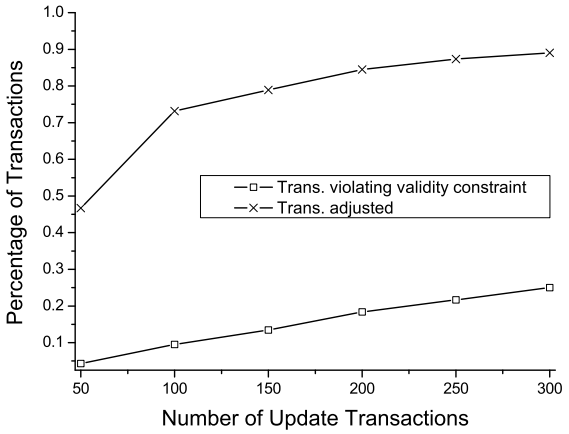

Figure 6.17: Pct. of adjusted transactions

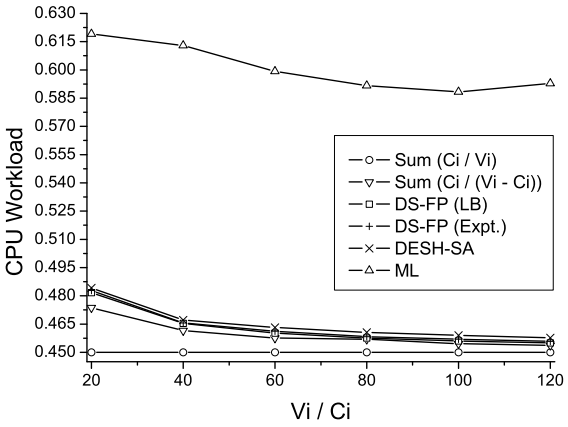

Figure 6.18: CPU util with fixed $\sum_{i=1}^{m} \frac{C_i}{V_i}$

workload for *DS-FP* should be higher than what is shown in the figure. Also note that the extra overhead resulting from *DESH-SA* is adjustable. For example, if the interval $[0, t_e]$ is further enlarged, the CPU workload of *DESH-SA* can be reduced, in which case it can become even closer to that of *DS-FP*.

Figure 6.14 shows the average number of adjusted jobs (N_{adjust}) obtained for each transaction in *DESH-SA*. The data is collected from 50 different transaction sets. It should be noted that the average number of adjusted jobs for each transaction is lower than 2, and it increases while the priority of a transaction decreases. Considering that such job adjustment produces a hyperperiod schedule that reduces on-line scheduling overhead to $O(1)$, the benefit of the adjustment is justified.

Next, we compare the hyperperiod length calculated from *DESH-SC* and *DESH-SA*. Figure 6.15 shows that the hyperperiod length for *DESH-SA* remains almost the same with the increase of the number of transactions in the system. But the hyperperiod length for *DESH-SC* increases rapidly and when the number of transactions in the system increases beyond 35, *DESH-SC* cannot find a hyperperiod. A shorter hyperperiod implies that less space overhead is needed for keeping scheduling information. This implies that *DESH-SA*

is also *space efficient*.

We also study the impact of $U_{\max}$, the hyperperiod utilization limit in *DESH-SC* (Alg. 12), on the resulting hyperperiod length. Figure 6.16 shows the relationship of the resulting hyperperiod length and $U_{\max}$. In the figure, $U_{\max}$ is varied as $1.02 \times$ *DS-FP*($Est.$), $1.06\times$*DS-FP*($Est.$), and $1.10\times$*DS-FP*($Est.$). The differences among the hyperperiod lengths are really small. A larger $U_{\max}$ results in a slightly shorter hyperperiod because *DESH-SC* terminates sooner with larger $U_{\max}$ in Alg. 12.

Figure 6.17 shows the percentage of transactions adjusted for *DESH-SA*. There are two percentage curves in the figure. The higher one is the percentage of transactions adjusted no matter whether they violate Eq. 6.37. Those transactions include all τ_i where $h \leq i \leq m$ (h is calculated at Line 3 in Alg. 13). The lower curve is the percentage of transactions which actually violate Eq. 6.37. The former should be higher than the latter because if the schedule of a transaction, e.g., τ_h, is adjusted, then the schedule of its lower-priority transactions will be adjusted even if a lower-priority one does not violate Eq. 6.37 before τ_h's adjustment.

As mentioned earlier, we also conducted experiments by varying $\frac{\mathcal{V}_i}{C_i}$ and fixing $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i}$ of the update transaction set at 45%. The results are depicted in Figure 6.18, which compares *ML*, *DS-FP*, *DESH-SA*, $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i}$ and $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$. As in Figure 6.13, the actual utilization for *DS-FP* is very close to the estimated average utilization $\overline{U}_{DS}$ (shown as *DS-FP*(Est.)). As explained in [92], $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$ is the CPU workload resulting from the possible maximum separation $\mathcal{V}_i - C_i$ satisfying the *validity constraint* for each transaction τ_i. It is the CPU lower bound ignoring transaction interference. It is observed in Figure 6.18 that the CPU workloads of*DS-FP* and *DESH-SA* are very close to that of $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$. The larger $\frac{\mathcal{V}_i}{C_i}$ is, the closer *DS-FP*, *DESH-SA* and $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$ are. This is because the probability of transaction interference decreases for *DS-FP* and *DESH-SA* when $\frac{\mathcal{V}_i}{C_i}$ becomes larger.

In summary, it is demonstrated in our experimental results that the CPU workloads produced by the *DESH* algorithms nearly match those produced by *DS-FP*. In addition,

DESH-SA outperforms *DESH-SC* in terms of time and space complexity. Thus, *DESH-SA* is a very efficient algorithm for reducing the *DS-FP* overhead.

6.4 *DS-FP* Schedulability Analysis

Although *DS-FP* can reduce the update workload, sacrificing the periodicity of updates poses great challenges for its schedulability analysis. One prominent method in classical schedulability analysis is based on the *critical instant test* [95]. A critical instant makes sense for periodic tasks by assuming synchronous update tasks, *i.e.*, all of the first jobs of update tasks are initiated at the same time. It has also been adopted for sporadic task sets by converting the minimum separation times to be the periods in the schedulability analysis. In a *DS-FP* schedule, however, the distance of the release times of two consecutive jobs from the same update transaction is not fixed. It is only proven so far that *DS-FP* could schedule any transaction set that is schedulable by *ML* [93]. However, the truth of the converse is still an open problem. This sufficient condition has seriously restricted the schedulability of a transaction set by *DS-FP*. Although experimentally it has been demonstrated that *DS-FP* outperforms *ML* significantly [93], the open theoretic question of whether there is any necessary and sufficient conditions to determine if a transaction set is schedulable by *DS-FP* (even if it is not schedulable by *ML*) is still unsolved.

In this section, we address the problem of finding necessary and sufficient conditions for the schedulability of *DS-FP*. In Section 6.4.1, we demonstrate by examples that transactions schedulable by *DS-FP* are not necessarily schedulable by *ML*, and introduce the concept of *pattern*. Section 6.4.2 proves the existence of a repeating *DS-FP* pattern in a valid *DS-FP* schedule. Properties of the *DS-FP* pattern are presented in Section 6.4.3. Section 6.4.4 presents an algorithm that searches for the *earliest* and *shortest* pattern in a *DS-FP* schedule. The search algorithm forms the basis for a schedulability test algorithm in Section 6.4.5. Finally Section 6.4.6 discusses *DS-FP* in continuous time systems.

From here on, it is assumed that transactions are studied in a discrete time system

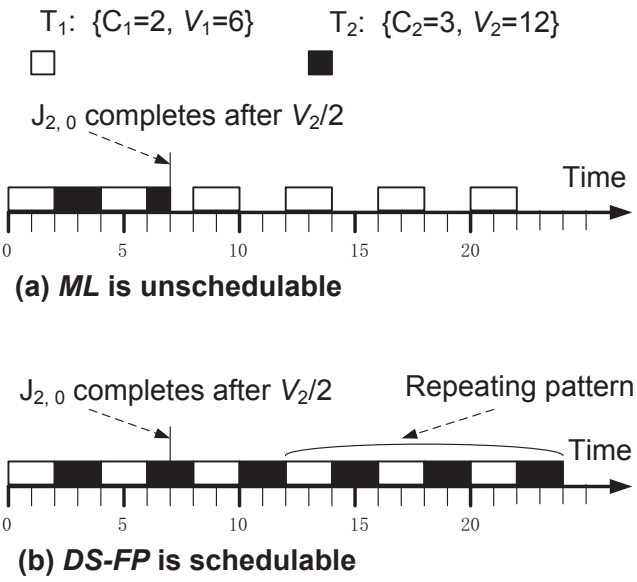

Figure 6.19: Two transactions that can be scheduled by *DS-FP* but not by *ML*

unless it is specified otherwise.

6.4.1 *DS-FP* Pattern

Theorem 6.2.2 states that *DS-FP* is at least as good as *ML* in terms of schedulability. That is, if $\mathcal{T}$ can be scheduled by *ML*, then it can also be scheduled by *DS-FP*. However, the converse statement is not true. This can be demonstrated in the following examples.

Example 6.4.1: Consider a set of two transactions $\{\tau_1, \tau_2\}$ with computation times 2 and 3, and validity intervals 6 and 12 respectively. Figure 6.19 (a) depicts a schedule of the transactions under *ML*. The first job of τ_2, $J_{2,0}$, completes at time 7, which is greater than $\frac{\mathcal{V}_2}{2} = 6$. Thus the set of transactions is not schedulable by *ML*.

Figure 6.19 (b) depicts a schedule of the transactions under *DS-FP*. The same transaction set is schedulable by *DS-FP* because the schedule pattern between time 12 and 24 repeats itself forever. □

DS-FP is better in Example 6.4.1 because *DS-FP* allows $J_{2,0}$ to be completed later than $\frac{\mathcal{V}_2}{2}$. There are also transaction sets in which for every transaction τ_i, $J_{i,0}$ is completed no later than $\frac{\mathcal{V}_i}{2}$ in *DS-FP*, and further more these transaction sets can be scheduled by *DS-FP* but not by *ML*. This is because *DS-FP* leaves spare processor time early on for being used by $J_{i,0}$s of lower priority transactions. The next example illustrates this point.

Example 6.4.2: Consider a set of three transactions $\{\tau_1, \tau_2, \tau_3\}$ with computation times 2, 3, 3, and validity intervals 6, 15, 47, respectively. Table 6.9 summarizes the release times and deadlines for the jobs in each transaction under *ML* and *DS-FP*. Figure 6.20 (a) depicts a schedule of the transactions under *ML*. The *ML* period and deadline for τ_2 are 8 and 7, respectively. The first job of τ_3, $J_{3,0}$, completes at time 24, which is greater than $\frac{\mathcal{V}_3}{2} = 23.5$. Thus the set of transactions is not schedulable by *ML*.

Figure 6.20 (b) depicts a schedule of the transactions under *DS-FP*. The same transaction set is schedulable by *DS-FP* because the schedule pattern between time 26 and 50 repeats itself forever. In this schedule $J_{3,0}$ completes at time 19, which is smaller than $\frac{\mathcal{V}_3}{2}$ (i.e., 23.5). This is because $J_{2,2}$ is scheduled later than that of *ML*. □

Note that *DS-FP* fully utilizes the processor in both examples. We could easily derive examples in which the processor idles once in a while. For example, in Figure 6.19 we could change C_2 to 2.5 and in Figure 6.20 we could change C_3 to 2.75. After both changes the transaction sets still cannot be scheduled by *ML* but can by *DS-FP*. Furthermore, we can scale up the numbers to make them all integers again.

We denote by a tuple $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$ the *DS-FP* schedule of length $\mathcal{P}_l$ starting from time $\mathcal{P}_s$. Let tuple $\mathcal{S}_\tau(t) = (d, e)$ denote the state of transaction τ at time t, where d is the distance to τ's last job release before time t, and e is the remaining outstanding execution time of τ at time t. In particular, $e = 0$ if τ's last job before t has already finished at time t. We denote by $\mathcal{S}_\mathcal{T}(t)$ the aggregated states of all transactions in $\mathcal{T}$ at time t, i.e., $\mathcal{S}_\mathcal{T}(t) = \{\mathcal{S}_\tau(t) \mid \tau \in \mathcal{T}\}$. Note once $\mathcal{S}_\mathcal{T}(t)$ is known, the *DS-FP* schedule from t onward can be determined.

	τ_1	τ_2		τ_3	
Job	*ML* /*DS-FP*	*ML*	*DS-FP*	*ML*	*DS-FP*
0	(0,2)	(0, 7)	(0, 7)	(0, 24)	(0, 19)
1	(4,6)	(8, 15)	(10, 15)		(26, 47)
2	(8,10)	(16, 23)	(19, 25)		(50, 73)
3	(12,14)	(24, 31)	(27, 34)		(74,97)
4	(16,18)	(32, 39)	(35, 42)		(98,121)
5	(20,22)	(40, 47)	(43, 50)		...
6	(24,26)	(48, 55)	(51,58)		...
7	(28,30)	(56, 63)	(59,66)		...
8	(32,34)	(64, 71)	(67,74)		...
9	(36,38)	(72, 79)	(75,82)		...
10	(40,42)	(80, 87)	(83,90)		...
11	(44,46)	(88, 95)	(91,98)		...
12	(48,50)	(96, 103)	(99,106)		...
13	(52,54)	...	...		...
14	(56,58)	...	...		...
15	(60,62)	...	...		...
16	(64,66)	...	...		...
17	(68,70)	...	...		...
18	(72,74)	...	...		...
19	(76,78)	...	...		...
20	(80,82)	...	...		...
21	(84,86)	...	...		...
22	(88,90)	...	...		...
23	(92,94)	...	...		...
24	(96,98)	...	...		...

Table 6.9: Release time and deadline comparison in Example 6.4.2

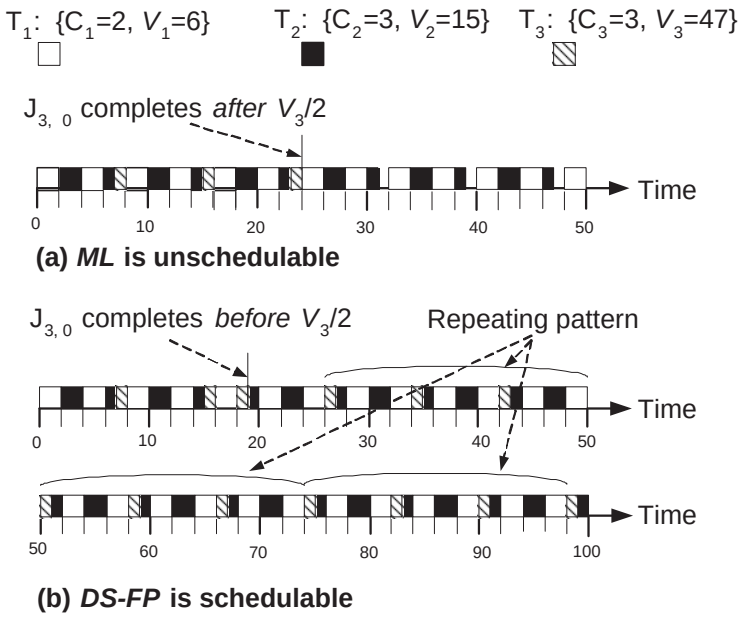

Figure 6.20: Three transactions that can be scheduled by *DS-FP* but not by *ML*

Definition 6.4.1: Given a transaction set $\mathcal{T}$, if (1) a *DS-FP* schedule repeats pattern $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$ forever in time interval $[\mathcal{P}_s + n\mathcal{P}_l, \mathcal{P}_s + (n+1)\mathcal{P}_l)$ $(n = 0, 1, 2, ..)$; and (2) $\mathcal{S}_\mathcal{T}(\mathcal{P}_s + n\mathcal{P}_l + t) = \mathcal{S}_\mathcal{T}(\mathcal{P}_s + (n+1)\mathcal{P}_l + t)$ $(t = 0, 1, 2, .., \mathcal{P}_l - 1)$, then $\mathcal{P}$ is a repeating *pattern* of $\mathcal{T}$'s *DS-FP* schedule.

Corollary 6.4.1: If $S_\mathcal{T}(t+s) = S_\mathcal{T}(t)$, then $(P_s = t, P_l = s)$ is a repeating pattern.

Proof. By definition of $S_\mathcal{T}(t)$, for any t, $S_\mathcal{T}(t+1)$ is fully determined based on $S_\mathcal{T}(t)$. So if $S_\mathcal{T}(t+s) = S_\mathcal{T}(t)$, we have $S_\mathcal{T}(t+s+1) = S_\mathcal{T}(t+1)$ and the same transaction is scheduled at time $t+s+1$ and $t+1$ if time t is not idle. Following the same argument, we have, for all $k > 0$, $S_\mathcal{T}(t+s+k) = S_\mathcal{T}(t+k)$ and thus $S_\mathcal{T}(t+s) = S_\mathcal{T}(t+s+s)$. This implies that $(P_s = t, P_l = s)$ is a repeating pattern.□

Note that we prove that a transaction set is schedulable by *DS-FP* by demonstrating that a repeating pattern occurs for the transaction set. The remaining questions are whether a repeating pattern always exists in a *DS-FP* schedule if the transaction set is schedulable,

and if so, how to find it. We answer those questions next.

6.4.2 *DS-FP* Pattern Analysis

In this subsection, we prove that for each transaction set which is schedulable by *DS-FP*, there always exists a repeating pattern. Note that *DS-FP* is not necessarily idle immediately before $\mathcal{P}_s$ for a pattern $\mathcal{P}$.

Example 6.4.3: Consider the same transaction set as in Example 6.4.1. $\mathcal{P} = (12, 24)$ is a repeating pattern because the schedule between time 12 and 24 repeats itself forever. Also $\mathcal{S}_{\mathcal{T}}(12) = \mathcal{S}_{\mathcal{T}}(24)$ because $\mathcal{S}_{\tau_1}(12) = \mathcal{S}_{\tau_1}(24) = (4, 0)$ and $\mathcal{S}_{\tau_2}(12) = \mathcal{S}_{\tau_2}(24) = (5, 0)$. However, although the schedule between time 6 and 10 does repeat itself, $\mathcal{P}' = (6, 4)$ is not a repeating pattern because $\mathcal{S}_{\mathcal{T}}(6) \neq \mathcal{S}_{\mathcal{T}}(10)$. □

As mentioned before, we study the *DS-FP* schedulability problem in a discrete time system. In order to prove that there exists a repeating pattern if a transaction set is schedulable by *DS-FP*, we shall first review the *Pigeonhole* Principle.

The Pigeonhole Principle [30]: If m pigeons occupy n pigeonholes and $m > n$, then at least one pigeonhole has two or more pigeons roosting in it.

In a discrete time system, a repeating pattern always exists for any successful *DS-FP* schedule because we know the fact that the execution times, validity intervals, and the number of transactions are all finite integers, and so an execution state can be defined that characterizes the progress of an execution in meeting the timing constraints for any particular time. Given infinite time, there must be a pattern repeating itself in the *DS-FP* schedule as the number of distinct execution states is finite. The following theorem states that a *DS-FP* schedule has a repeating pattern that must occur at least once in a bounded time interval.

Theorem 6.4.1: Given an update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ $(1 \leq i \leq m)$, if it can be scheduled by *DS-FP* in the bounded time interval $[0, (\mathcal{V}_m - C_m) + \Pi_{i=1}^{m}(\mathcal{V}_i - C_i + 1) - 1]$,

then the *DS-FP* schedule has a *repeating pattern* that must occur at least once in the bounded time interval $[\mathcal{V}_m - C_m, (\mathcal{V}_m - C_m) + \Pi_{i=1}^{m}(\mathcal{V}_i - C_i + 1) - 1]$.

Proof. The theorem can be proved by the following two claims using induction:

1. There is a *pattern* for τ_1 in the interval $[\mathcal{V}_m - C_m, \mathcal{V}_m - C_m + \mathcal{V}_1 - C_1]$ which repeats itself in the *DS-FP* schedule.

2. For any k, $1 \leq k < m$, if there is a *pattern* for the schedule of τ_1, .., τ_k in the interval $[\mathcal{V}_m - C_m, \mathcal{V}_m - C_m + \Pi_{i=1}^{k}(\mathcal{V}_i - C_i + 1) - 1]$ which repeats itself in the *DS-FP* schedule, then there is a *pattern* for τ_1, .., τ_k, τ_{k+1} in the interval $[\mathcal{V}_m - C_m, \mathcal{V}_m - C_m + \Pi_{i=1}^{k+1}(\mathcal{V}_i - C_i + 1) - 1]$ which repeats itself in the same *DS-FP* schedule.

The first claim is obvious because there is a repeating pattern of length $(\mathcal{V}_1 - C_1)$ repeating itself from time 0. As a matter of fact, any schedule of length $(\mathcal{V}_1 - C_1)$ is a repeating pattern, thus the schedule of length $(\mathcal{V}_1 - C_1)$ from time $\mathcal{V}_m - C_m$ must repeat itself. Note that the theorem does not require that the first instance of the repeating pattern in the interval $[\mathcal{V}_m - C_m, \mathcal{V}_m - C_m + \Pi_{i=1}^{m}(\mathcal{V}_i - C_i + 1) - 1]$ starts exactly on time $\mathcal{V}_m - C_m$.

Now let us prove the second claim. We shall rely on the *Pigeonhole* Principle to identify two time points in two instances of the recurring pattern for transactions τ_1, .., τ_k such that the following two conditions are satisfied: 1) the two time points are τ_{k+1}'s release times; 2) the two time points have the same offsets within their patterns. If such two time points are identified, then the schedule of τ_1, .., τ_k, τ_{k+1} between those two time points is a repeating pattern repeating itself thereafter. This is because the schedules after those two time points are identical for transactions τ_1, .., τ_k, thus it is also identical for τ_{k+1}.

Suppose that the repeating pattern for τ_1, .., τ_k starting from time t has length L. We have $t \geq (\mathcal{V}_m - C_m)$ and $L \leq \Pi_{i=1}^{k}(\mathcal{V}_i - C_i + 1) - 1$. There are two cases, $L \geq (\mathcal{V}_{k+1} - C_{k+1})$ and $L < (\mathcal{V}_{k+1} - C_{k+1})$.

Case I: Suppose $L \geq (\mathcal{V}_{k+1} - C_{k+1})$. In every recurring instance of the repeating pattern of length L starting from time t, there is at least one job of τ_{k+1} since $L \geq (\mathcal{V}_{k+1} - C_{k+1})$.

Let us examine the last τ_{k+1} job in each recurring instance of the pattern. Denote d to be the distance from its release time to the end of the pattern. The length of d cannot exceed $(\mathcal{V}_{k+1} - C_{k+1})$, otherwise there must be another job afterwards in the pattern in order to satisfy τ_{k+1}'s validity constraint. Since $d > 0$, it can be one of $(\mathcal{V}_{k+1} - C_{k+1})$ possible values (pigeonholes). Let us look at τ_{k+1}'s last job (pigeon) in each of the $(\mathcal{V}_{k+1} - C_{k+1} + 1)$ recurring patterns since time t. It follows from the *Pigeonhole* Principle that there must be two jobs' release time at the same offset within their corresponding pattern instances. Denote t_1 and t_2 to be the two job release times, and $t_1 < t_2$. We then have a repeating pattern that must occur at least once in the interval $[t_1, t_2]$ repeating itself thereafter for transactions τ_1,.., τ_k, τ_{k+1}. Since $t_1 \geq t \geq (\mathcal{V}_m - C_m)$ and $t_2 < (\mathcal{V}_m - C_m) + L(\mathcal{V}_{k+1} - C_{k+1} + 1) < (\mathcal{V}_m - C_m) + \Pi_{i=1}^{k+1}(\mathcal{V}_i - C_i + 1) - 1$, we have proved the first case.

Case II: Suppose $L < (\mathcal{V}_{k+1} - C_{k+1})$. Denote $J_{k+1,w}$ to be the first τ_{k+1}'s job that executes after time t. $J_{k+1,w}$ and all its subsequent jobs appear in some instance (not necessarily the same instance) of the pattern. There are only L possible offsets (pigeonholes) within a pattern for τ_{k+1}'s jobs to start. Let us look at the first $(L + 1)$ jobs (pigeon) of τ_{k+1}, i.e., $J_{k+1,w}$ through $J_{k+1,w+L}$. It follows from the *Pigeonhole* Principle that there must be two jobs starting at the same offset within their corresponding pattern instances. Denote t_1 and t_2 to be the two job release times in *DS-FP*, and $t_1 < t_2$. We then have a repeating pattern that must occur at least once in the interval $[t_1, t_2]$ repeating itself for transactions τ_1,.., τ_k, τ_{k+1}. Since $t_1 \geq t \geq (\mathcal{V}_m - C_m)$ and $t_2 < (\mathcal{V}_m - C_m) + (L + 1)(\mathcal{V}_{k+1} - C_{k+1}) < (\mathcal{V}_m - C_m) + \Pi_{i=1}^{k+1}(\mathcal{V}_i - C_i + 1) - 1$, we have proved the second case.

Based on the above two claims, the theorem is proved. □

According to the proof of Theorem 6.4.1, if a transaction set can be scheduled by *DS-FP* in the interval $[0, (\mathcal{V}_m - C_m) + \Pi_{i=1}^{m}(\mathcal{V}_i - C_i + 1) - 1]$, then it is schedulable by *DS-FP* because a repeating pattern appearing in the interval repeats itself forever. Thus we have the following corollary.

Corollary 6.4.2: An update transaction set $\mathcal{T}$ can be scheduled by *DS-FP* if and only if it can be scheduled by *DS-FP* in the interval $[0, (\mathcal{V}_m - C_m) + \Pi_{i=1}^{m}(\mathcal{V}_i - C_i + 1) - 1]$.

6.4.3 *DS-FP* Pattern Properties

Theorem 6.4.1 proves the existence of a repeating pattern for a given *DS-FP* schedule. This subsection further studies the properties of the *DS-FP* pattern.

Given a repeating pattern $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$, the following corollary follows directly from the fact that all transactions have the same states at times $\mathcal{P}_s + t$ and $\mathcal{P}_s + t + \mathcal{P}_l$ for $t > 0$, i.e., $\mathcal{S}_{\mathcal{T}}(\mathcal{P}_s + t) = \mathcal{S}_{\mathcal{T}}(\mathcal{P}_s + t + \mathcal{P}_l)$.

Corollary 6.4.3: If $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$ is a pattern repeating itself from time $\mathcal{P}_s$, then $(\mathcal{P}_s + t, \mathcal{P}_l)$ $(t > 0)$ is also a pattern repeating itself from time $\mathcal{P}_s + t$.

We now prove the next lemma.

Lemma 6.4.1: Given all the patterns $\mathcal{P}$, $\mathcal{P}'$, ... of a *DS-FP* schedule for transaction set $\mathcal{T}$, let $\mathcal{P}$ be a pattern with the minimum length among all patterns, i.e., $\mathcal{P}_l \leq \mathcal{P}'_l$ for any other pattern $\mathcal{P}'_l$. Then $\mathcal{P}'_l$ is a multiple of $\mathcal{P}_l$, i.e., $\mathcal{P}'_l = N\mathcal{P}_l$ where N is a positive integer.

Proof. Let $t_1 = \mathcal{P}'_s + \mathcal{P}'_l * n_1$ ($n_1 > 0$ is an integer) such that $t_1 > \mathcal{P}_s$. Both $(t_1, \mathcal{P}_l)$ and $(t_1, \mathcal{P}'_l)$ are patterns.

We prove the lemma by contradiction. Suppose $\mathcal{P}'_l$ is not a multiple of $\mathcal{P}_l$ and $\mathcal{P}'_l = \mathcal{P}_l * r - s$, $r \geq 2$, and $0 < s < \mathcal{P}_l$. We have the state $\mathcal{S}_{\mathcal{T}}((t_1 + \mathcal{P}'_l) + s) = \mathcal{S}_{\mathcal{T}}(t_1 + \mathcal{P}_l * r) = \mathcal{S}_{\mathcal{T}}(t_1) = \mathcal{S}_{\mathcal{T}}(t_1 + \mathcal{P}'_l)$. It follows Corollary 6.4.1 that the pattern $(t_1 + \mathcal{P}'_l, s)$ repeats itself from time $(t_1 + \mathcal{P}'_l)$ with length s. As $0 < s < \mathcal{P}_l$, this contradicts the fact that $\mathcal{P}_l$ is the minimum length among all repeating patterns. So $\mathcal{P}'_l$ must be a multiple of $\mathcal{P}_l$. □

In the proof of Lemma 6.4.1, since $\mathcal{P}'_l$ is a multiple of $\mathcal{P}_l$, $(\mathcal{P}'_s, \mathcal{P}_l)$ must also be a pattern.

Corollary 6.4.4: Given all the patterns of a *DS-FP* schedule, let $\mathcal{P}$ be a pattern with the minimum $\mathcal{P}_l$. For any other pattern $\mathcal{P}'$, $(\mathcal{P}'_s, \mathcal{P}_l)$ is also a repeating pattern.

Lemma 6.4.1 and Corollary 6.4.4 imply that there exists a shortest pattern $\mathcal{P}$ that is also the earliest. Any other pattern $\mathcal{P}'$ could be derived from $\mathcal{P}$. $\mathcal{P}'$ could be of the same length but with some offset from a $\mathcal{P}$'s repeat; $\mathcal{P}'$ could be a multiple of $\mathcal{P}$'s repeats; or $\mathcal{P}'$ could be a multiple of $\mathcal{P}$'s repeats with some offset.

Lemma 6.4.2: If $\mathcal{P}'$ and $\mathcal{P}''$ are two different repeating patterns of a *DS-FP* schedule, then $(\mathcal{P}'_s, \mathcal{P}''_l)$ and $(\mathcal{P}''_s, \mathcal{P}'_l)$ are also repeating patterns.

Proof. Let $\mathcal{P}$ be the shortest and earliest pattern. According to Lemma 6.4.1, $\mathcal{P}''_l$ is a multiple of $\mathcal{P}_l$. According to Corollary 6.4.4, $(\mathcal{P}'_s, \mathcal{P}_l)$ is also a repeating pattern. Because $(\mathcal{P}'_s, \mathcal{P}_l)$ is a pattern and $\mathcal{P}''_l$ is a multiple of $\mathcal{P}_l$, $(\mathcal{P}'_s, \mathcal{P}''_l)$ is also a repeating pattern.
By the same argument, $(\mathcal{P}''_s, \mathcal{P}'_l)$ is a repeating pattern. □

Given transaction set $\mathcal{T}$ of size m, we call $\mathcal{P}^i$ a pattern of the first i ($1 \leq i \leq m$) highest priority transactions (τ_1, .., τ_i) by ignoring all other lower priority transactions τ_{i+1},...,τ_m in the schedule. In other words, $\mathcal{P}^i$ is a pattern of the transaction set consisting of only the first i highest priority transactions.

Lemma 6.4.3: If $\mathcal{P}^i$ is the shortest and earliest pattern of the first i ($1 \leq i < m$) highest priority transactions, and $\mathcal{P}^{i+1}$ is the shortest and earliest pattern of the first $i + 1$ highest priority transactions, then

1.$\mathcal{P}^i_s \leq \mathcal{P}^{i+1}_s$.

2.$\mathcal{P}^{i+1}_l$ is a multiple of $\mathcal{P}^i_l$.

Proof. By ignoring the schedule of τ_{i+1} in $\mathcal{P}^{i+1}$, $\mathcal{P}^{i+1}$ is also a repeating pattern of the first i highest priority transactions. By definition, $\mathcal{P}^i_s \leq \mathcal{P}^{i+1}_s$. By Lemma 6.4.1, $\mathcal{P}^{i+1}_l$ is a multiple of $\mathcal{P}^i_l$. □

***DS-FP* patterns in general cases.** Please note that we assume the worst-case execution times for all jobs in our *DS-FP* pattern analysis. However, this assumption does not always hold. Lemma 6.4.4 states that in the general case where a job's actual execution time can

be less than its worst-case execution time, the *DS-FP* pattern can still be kept if the *DS-FP* scheduler still assigns the worst-case execution time to the job. In such cases, the CPU may idle after the job's completion until the job's assigned time slots expire.

Lemma 6.4.4: Given an update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ ($1 \leq i \leq m$), if it can be scheduled by *DS-FP* with worst-case execution times, then it can also be scheduled by *DS-FP* in general cases and each job $J_{i,j}$ in both schedules can have the same release time $r_{i,j}$ and deadline $d_{i,j}$.

Proof. Let $c_{i,j}$ be the actual execution time of $J_{i,j}$ in general cases and we have $c_{i,j} \leq C_i$. Let S be a feasible *DS-FP* schedule with the worst-case execution times of all tasks. Let us keep $J_{i,j}$'s release time $r_{i,j}$ and deadline $d_{i,j}$ unchanged and replace $J_{i,j}$'s worst-case execution time C_i in S with its corresponding actual execution time $c_{i,j}$. Since $c_{i,j} \leq C_i$, $J_{i,j}$ must be schedulable after the replacement. If we replace every $J_{i,j}$'s worst-case execution time C_i with its actual execution time $c_{i,j}$, we have a feasible schedule S' for the general case execution times. □

6.4.4 *DS-FP* Pattern Search Algorithm

Corollary 6.4.2 forms a basis for the schedulability test of *DS-FP*. But the length of the interval in Corollary 6.4.2 is $O(\Pi_{i=1}^{m}\mathcal{V}_i)$, and it does not take into consideration the slots occupied by $C_i (1 \leq i \leq m)$. We now present an improved upper bound estimation of the pattern length by restricting the possible pigeonholes only to the idle slots in the *DS-FP* schedules.

Given a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$, we denote by $\overline{\mathcal{P}}^i$ the upper bound length of the pattern $\mathcal{P}^i$ formed by transactions $\tau_1, \tau_2, \ldots, \tau_i$, and I^{i-1} the number of idle slots in $\mathcal{P}^{i-1}$. Consider $I^{i-1} + 1$ consecutive jobs of τ_i following $\mathcal{P}_s^{i-1}$, there are two notable facts about those τ_i jobs:

1. there exist two jobs starting at the same offset within their corresponding pattern $\mathcal{P}^{i-1}$

instances according to the pigeonhole principle; and

2. the separation between any two consecutive jobs of τ_i can not exceed $\mathcal{V}_i - C_i$.

The schedule between the request time points of the two jobs in the first fact forms a repeating pattern. Thus, $\overline{\mathcal{P}}^i (2 \leq i \leq m)$ can be estimated as follows.

$$\overline{\mathcal{P}}^i = (I^{i-1} + 1) \cdot (\mathcal{V}_i - C_i) \tag{6.45}$$

Following Eq. 6.45, the initial conditions $\overline{\mathcal{P}}^1 = \mathcal{V}_1 - C_1$ and $I^1 = \mathcal{V}_1 - 2 \cdot C_1$, the upper bound of the pattern length $\overline{\mathcal{P}}^i$ for transactions τ_i $(1 \leq i \leq m)$ can be estimated iteratively from high to low priority transactions, which helps improve the efficiency of our pattern search Alg. 14 and 15. Note that the computation of upper bound $\overline{\mathcal{P}}^i$ only takes into account the idle slots in $\mathcal{P}^{i-1}$ (i.e., I^{i-1}) while not $\mathcal{P}_l^{i-1}$. This significantly reduces the length of the time interval for finding its pattern $\mathcal{P}^i$.

Our pattern search algorithm follows the idea in Theorem 6.4.1. It searches for the pattern of the first i $(2 \leq i \leq m)$ highest priority transactions based on the repeating pattern of the first $i - 1$ highest priority transactions. After the algorithm completes for the lowest priority transaction, it returns the pattern for the transaction set.

Alg. 14 invokes Alg. 15 whose input is the pattern of the first $i-1$ $(1 < i \leq m)$ highest priority transactions, and output is the pattern of the first i highest priority transactions. Alg. 15 scans the *DS-FP* schedule for the jobs of the i^{th} highest priority transaction to find the first two jobs such that each starts at the same offset within its corresponding input pattern $\mathcal{P}^{i-1}$. The schedule between these two release times forms the output pattern for the first i highest priority transactions.

Note that in Alg. 15, τ_i can only be scheduled in the idle slots of the input pattern $\mathcal{P}^{i-1}$. According to the *Pigeonhole* Principle, Alg. 15 does not need to examine more jobs than the number of idle slots plus 1 in $\mathcal{P}^{i-1}$. In other words, the while loop of Line 7 in Alg. 15 does not need to loop more than the number of idle slots in $\mathcal{P}^{i-1}$ plus 1. Thus the

condition at Line 10 can be true at least once before the while loop beginning from Line 7 ends.

Line 11 in Alg. 15 produces the shortest pattern starting from the release time of one of τ_i's jobs. Given a job $\tau_{i,j+k}$ that satisfies condition $((r_{i,j+k} - r) \,\%\, \mathcal{P}_l^{i-1} = 0)$ at Line 10, the while loop at Line 13 cannot run for more than $\mathcal{V}_i - C_i$ times. Otherwise the end of the found pattern must have hit the time point $r_{i,j+k-1}$ and the beginning of the pattern must also be a τ_i's request time because the beginning and the end of a pattern have the same state. However, this pattern must have already satisfied the condition on line 7 during the previous while loop of line 4 and must have been returned by the algorithm. Also note that the while loop cannot move back to τ_i's first job $J_{i,0}$ ($i \geq 2$) because the release time of $J_{i,0}$ (i.e., time 0) is not the beginning time of $J_{i,0}$'s execution in the *DS-FP* algorithm.

Theorem 6.4.2: $\mathcal{P}^m$ returned by Alg. 14 is the shortest and earliest pattern.

Proof. We shall prove that if the input to Alg. 15 is the shortest and earliest pattern, so is the output.

We first prove that Alg. 15 returns a pattern. Alg. 15 returns only when the condition at Line 10 is true. The condition implies that r and $r_{i,j+k}$ are of the same offsets within their respective input patterns. So $\mathcal{P}^i$ derived at Line 11 is a pattern for transactions τ_1, .., τ_{i-1}, τ_i. Furthermore, the condition of Line 13 guarantees that $\mathcal{P}^i$ remains to be a pattern when it is shifted along the time line.

We then prove that the returned $\mathcal{P}^i$ is the shortest. Let us examine $\mathcal{P}^i$ produced at Line 11. Assume that the shortest pattern is of length L and $L \leq \mathcal{P}_l^i$, then according to Corollary 6.4.4 $(\mathcal{P}_s^i, L)$ must be a pattern. The algorithm indicates that τ_i must have a job J released at time $\mathcal{P}_s^i + L$, which is earlier than or equal to $\mathcal{P}_s^i + \mathcal{P}_l^i$. According to Lemma 6.4.3, L is a multiple of $\mathcal{P}_l^{i-1}$. This means that J satisfies the condition at Line 10. Since $(\mathcal{P}_s^i, L)$ is the shortest, J should be the first examined job that satisfies the condition. In other words, $L = \mathcal{P}_l^i$. Finally, since $\mathcal{P}^i$ at Line 11 is the first pattern that starts with a τ_i's release time,

the while loop at Line 13 guarantees that the returned $\mathcal{P}^i$ is the earliest pattern for the first i highest priority transactions.

We have now proved that if the input to Alg. 15 is the shortest and earliest pattern, so is the output. We also know that Line 4 in Alg. 14 assigns the shortest and earliest pattern for τ_1. By induction, $\mathcal{P}^m$ returned by Alg. 14 is the shortest and earliest pattern of the transaction set. □

Alg 14 SearchPattern

```
1: Input: A successful DS-FP schedule.
2: Output: The earliest and shortest pattern P^m.
3:
4: P^1 ← (0, V_1 − C_1); {// Pattern of the first transaction.}
5: for i = 2 to m do
6:    {// Find the pattern when adding the next transaction.}
7:    P^i ← SearchNextTask(i, P^{i−1});
8: end for
9: return P^m;
```

Alg. 14 has time complexity $O(m(\Pi_{i=1}^{m}\mathcal{V}_i)^2)$. However, it can be further improved to $O(m\Pi_{i=1}^{m}\mathcal{V}_i)$ if an array of size $O(\mathcal{P}_l^{i-1})$ can be used when searching for pattern $\mathcal{P}^i$. We simply walk through the job requests of τ_i. For each job, we save its index number in the array entry where the entry index is equal to this job's relative offset in its corresponding $\mathcal{P}^{i-1}$ instance. If the array entry already has saved a job index, then these two jobs form a pattern. Thus the complexity is $O(\mathcal{P}_l^i + \mathcal{V}_i \times \mathcal{P}_l^i) = O(\mathcal{V}_i \times \mathcal{P}_l^i) = O(\Pi_{i=1}^{m}\mathcal{V}_i)$. If Alg. 15 is implemented this way, the complexity of Alg. 14 will be $O(m\Pi_{i=1}^{m}\mathcal{V}_i)$.

Note that the while loop at Line 13 is only executed once although it is within the two outer loops. It loops at most $\mathcal{V}_i - C_i$ times. Thus it is ignored in the complexity calculation.

Alg 15 SearchNextTask

Input: Pattern $\mathcal{P}^{i-1}$ of transactions τ_1, .., τ_{i-1}.
Output: Pattern $\mathcal{P}^{i}$ of transactions τ_1, .., τ_{i-1}, τ_i.

$k \leftarrow 1$;
$r_{i,j} \leftarrow$ first τ_i release time after $\mathcal{P}_s^{i-1}$;
$maxL \leftarrow 1 + I^{i-1}$;
while $(k < maxL)$ **do**
  $k \leftarrow k + 1$;
  for $r = r_{i,j}$ to $r_{i,j+k-1}$ **do**
    if $((r_{i,j+k} - r)$ % $\mathcal{P}_l^{i-1} = 0)$ {// Find the shortest pattern} **then**
      $\mathcal{P}^i \leftarrow (r, r_{i,j+k} - r)$;
      {// The next loop finds the earliest pattern}
      while $\mathcal{S}_{\mathcal{T}}(\mathcal{P}_s^i - 1) = \mathcal{S}_{\mathcal{T}}(\mathcal{P}_s^i - 1 + \mathcal{P}_l^i)$ **do**
        $\mathcal{P}^i \leftarrow (\mathcal{P}_s^i - 1, \mathcal{P}_l^i)$;
      end while
      return $\mathcal{P}^i$;
    end if
  end for
end while
return **No pattern found**;

6.4.5 *DS-FP* Schedulability Test Algorithm

Alg. 14 also implies a schedulability test algorithm. The algorithm begins the schedulability test with τ_1. Given a subset of transactions τ_1, .., τ_{i-1} ($1 < i \leq m$) that has been tested, the algorithm test transaction τ_i by adding the transaction to the subset until an added transaction is not schedulable or a pattern for all transactions is found. Given transaction τ_i, the algorithm schedules it along with the schedule of the higher priority transactions τ_1, .., τ_{i-1}, for which a pattern has already been found. Alg. 16 and Alg. 17 are modified versions of Alg. 14 and Alg. 15 for the schedulability test, respectively.

If Alg. 16 returns TRUE, it also produces the shortest pattern and the *DS-FP* schedule. The following example illustrates how the algorithm works.

Example 6.4.4: Consider a set of three transactions $\{\tau_1, \tau_2, \tau_3\}$ with computation times 1, 1, 2, and validity intervals 3, 7, 14, respectively. It is not schedulable by *ML* because τ_3 is finished by 8, which is more than $\frac{\mathcal{V}_3}{2} = 7$. Now we test whether it can be scheduled by *DS-FP* or not.

Figure 6.21 (a) corresponds to Line 4 of Alg. 16. It shows the pattern of τ_1.

Figure 6.21 (b) depicts the result of invoking Alg. 17 for τ_2. There is only one idle time slot in $\{\tau_1\}$'s pattern, so the release times of two consecutive jobs $J_{2,1}$ and $J_{2,2}$ after $\mathcal{P}_s^1 = 0$ forms a pattern $\mathcal{P}^2 = (5, 6)$.

Figure 6.21 (c) depicts the result of invoking Alg. 17 for τ_3. There are two idle time slots in $\{\tau_1, \tau_2\}$'s pattern $\mathcal{P}^2 = (5, 6)$, and the algorithm examines three consecutive jobs $J_{3,1}$, $J_{3,2}$, and $J_{3,3}$ after $\mathcal{P}_s^2 = 5$ to find an output pattern $\mathcal{P}^3 = (9, 18)$. Note that $r_{3,1}$ has an offset 4 within the pattern $\mathcal{P}^2 = (5, 6)$, while $r_{3,2}$ has an offset 2 within its corresponding pattern $\mathcal{P}^2 = (17, 6)$, and $r_{3,3}$ has an offset 4 within its corresponding pattern $\mathcal{P}^2 = (23, 6)$. The offset of $r_{3,3}$ matches that of $r_{3,1}$. So Alg. 17 goes to Line 19, and Alg. 16 returns that the transaction set is schedulable.

The shortest and earliest pattern for $\mathcal{P}^3$ is $(8, 18)$, one time unit earlier than the starting time of $\mathcal{P}^3$ returned from Alg. 16. The earliest pattern $\mathcal{P}^3 = (8, 18)$ can be returned

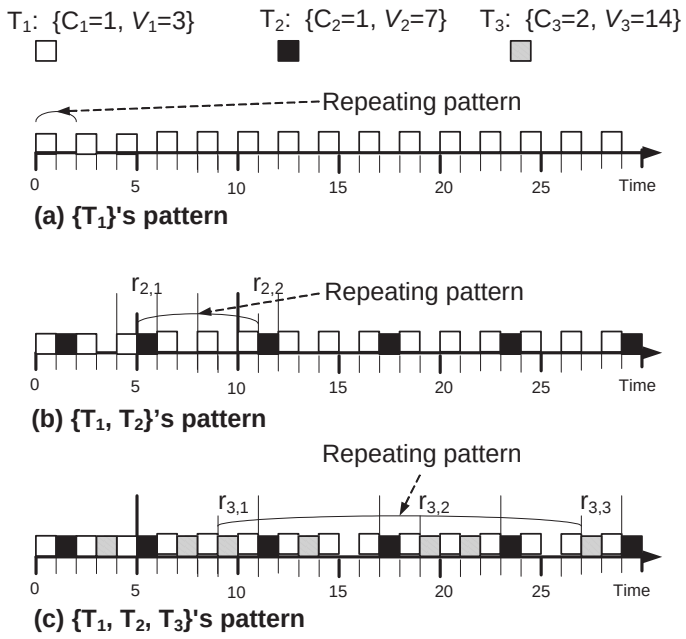

Figure 6.21: Illustration of the schedulability test algorithm

from Alg. 14. □

Given a *DS-FP* schedule, there always exists a repeating pattern and our schedulability test algorithm can be applied. However, the space and time complexity of the algorithm is high. The question remains if there is a better schedulability test that is more efficient.

6.4.6 *DS-FP* in Continuous Time Systems

So far we assume discrete time systems for *DS-FP*. Now we move on to the schedulability discussions of *DS-FP* in continuous time systems. Given a *DS-FP* schedule, it can be proved that a repeating pattern still exists if only rational numbers are considered for transaction parameters (i.e., validity intervals and execution times). Denote l to be the least common multiple of all the denominators of all those rational numbers. If we measure time in the unit of $\frac{1}{l}$, then we again have an integer problem which has a pattern for a successful

Alg 16 SchedulabilityTest

1: Input: A transaction set $\mathcal{T}$.
2: Output: Whether $\mathcal{T}$ is schedulable.
3:
4: $\mathcal{P}^1 \leftarrow (0, \mathcal{V}_1 - C_1)$; {// Pattern of the first transaction}
5: **for** $i = 2$ to m **do**
6: **if** (**TestNextTask**$(i, \mathcal{P}^{i-1})$ = FALSE) **then**
7: return $\mathcal{T}$ is unschedulable;
8: **end if**
9: **end for**
10: return $\mathcal{T}$ is schedulable;

Alg 17 TestNextTask

1: Input: Pattern $\mathcal{P}^{i-1}$ of transactions τ_1, .., τ_{i-1}.
2: Output: returns TRUE and pattern $\mathcal{P}^i$ of transactions τ_1, .., τ_{i-1}, τ_i if a pattern of those transactions exists. Otherwise, returns FALSE.
3:
4: Schedule up to, including τ_i's first request after $\mathcal{P}_s^{i-1}$;
5: **if** (Line 4 fails) **then**
6: return FALSE;
7: **end if**
8: $r_{i,j} \leftarrow \tau_i$'s first release time since $\mathcal{P}_s^{i-1}$;
9: $k \leftarrow 1$;
10: $maxL \leftarrow 1 + I^{i-1}$;
11: **while** $(k < maxL)$ **do**
12: $k \leftarrow k + 1$;
13: Schedule $r_{i,j+k}$;
14: **if** (Line 13 fails) **then**
15: return FALSE;
16: **end if**
17: **for** $r = r_{i,j}$ to $r_{i,j+k-1}$ **do**
18: **if** $((r_{i,j+k} - r)$ % $\mathcal{P}_l^{i-1} = 0)$ {// Found the shortest pattern.} **then**
19: $\mathcal{P}^i \leftarrow (r, r_{i,j+k} - r)$;
20: return TRUE;
21: **end if**
22: **end for**
23: **end while**

DS-FP schedule. This schedule is the same as the one that only has transaction parameters with original rational numbers although their granularities are different.

However, if execution times or validity intervals can be real numbers, it may not be possible to identify such a repeating pattern in a *DS-FP* schedule. We shall illustrate this with the following example.

Example 6.4.5: Consider a set of two transactions $\{\tau_1, \tau_2\}$ with computation times 1 and $1 + d$, and validity intervals 5 and 9 respectively. Suppose that d is an infinitely small real number. Figure 6.22(a) depicts a schedule of the transaction set under *DS-FP*. Let i to be the largest integer such that $3 - i \times d > 1$, i.e., $i = \lfloor \frac{2}{d} \rfloor$. $r_{2,1}, r_{2,2}, ..., r_{2,i}$ occur in every other repeating pattern of τ_1. In addition, $\forall k, (1 \leq k \leq i)$, the offset of $r_{2,k}$ within τ_1's pattern $\mathcal{P}$ is $3 - k \times d$. There exists no pattern for τ_2's first i jobs. Hence there exists no pattern from time 0 to $t = 2\mathcal{P}_l^i = 8\lfloor \frac{2}{d} \rfloor$. Time t can be arbitrarily large if d is infinitely small. In other words, if execution time C_2 of τ_2 is a real number infinitely close to 1, there exists no repeating pattern for the *DS-FP* schedule. Note that the transaction set has finite number of transactions, and finite values for execution times and validity intervals.

We can also prove that the transaction set is schedulable by *DS-FP* using induction. We know $J_{2,0}$ and $J_{2,1}$ are schedulable. We can easily prove that if $J_{2,i}, i > 0$ is schedulable, so is $J_{2,i+1}$. Another proof follows from Theorem 6.2.2 because the transaction set is obviously schedulable by *ML*. □

Our observation from Example 6.4.5 is the following: given an arbitrarily large time t ($t \to +\infty$), there always exists a transaction set with finite number of transactions and finite real number parameters that has a successful *DS-FP* schedule without any repeating pattern that must occur at least once in the interval $[0, t]$. However, there also exist transaction sets with finite number of transactions and finite real number parameters that have successful *DS-FP* schedules with repeating patterns, which is illustrated by the following example.

Example 6.4.6: Define two real numbers $p = \pi = 3.14159...$ and $e = 2.71828....$ Consider a set of two transactions $\{\tau_1, \tau_2\}$ with computation times p, 1, and validity intervals $3p$,

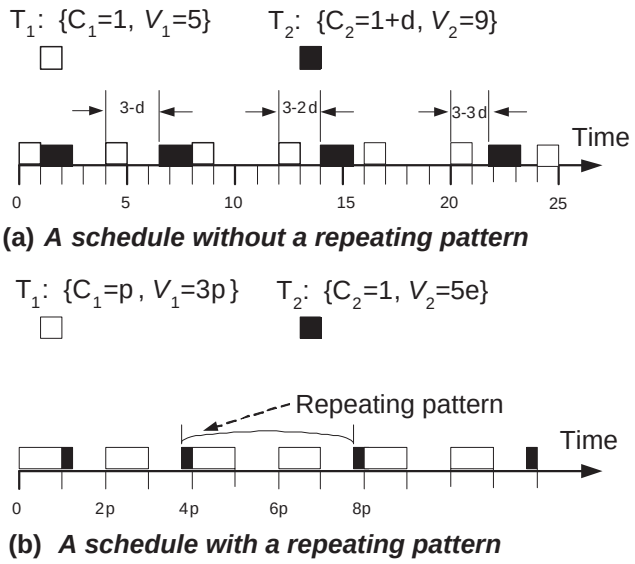

Figure 6.22: *DS-FP* schedules for transaction sets with real number parameters

5e, respectively. Figure 6.22 (b) depicts the *DS-FP* schedule of the transaction set with a repeating pattern. □

6.4.7 Performance Evaluation

In this section, we summarize the important results from our simulation studies on the schedulability of *DS-FP*. In the experiments, we compare their performance with *ML* as a baseline for reference.

Simulation Model and Parameters

A summary of the parameters and default settings used in experiments are presented in Table 6.10 and Table 6.11. The baseline values for the parameters are scaled down from those used in Section 6.2.5 for the effective demonstration of the *DS-FP* schedulability test. Three classes of parameters are defined: system parameters, update transaction parameters and application transaction parameters. For system configurations, we consider a single CPU,

Parameter Class	Parameters	Meaning
System	N_{CPU}	No. of CPU
	N_T	No. of real-time data objects
	$\mathcal{V}_i$	Validity interval of X_i
Update Transactions	C_i	CPU time for updating X_i
	Length	No. of data to update
Application Transactions	CPU Time	CPU time per data access
	Arrival Rate	Transaction arrival rate
	Slack Factor	Transaction slack factor

Table 6.10: Experimental parameters

Parameter Class	Parameters	Meaning
System	N_{CPU}	1
	N_T	[1, 15]
	$\mathcal{V}_i(ms)$	[20, 200]
Update Transactions	$C_i(ms)$	[1, 10]
	Length	1
Application Transactions	CPU Time (ms)	[1, 5]
	Arrival Rate	[20, 50]
	Slack Factor	8

Table 6.11: Experimental settings

main memory based RTDBS. The number of real-time data objects varies from 1 to 15 and the validity interval of each real-time data object is uniformly distributed between 20 and 200 ms. Each update transaction updates one real-time data object. The worst-case CPU execution time per update is 10 ms and the actual execution time is uniformly distributed between 1 and 10 ms. For the application transaction, the worst-case CPU execution time per query is 5 ms. We assume that its actual execution time is uniformly distributed between 1 and 5 ms and its arrival rate varies from 20 to 50 per second.

Following the definition of [97], we define the *density factor* of a set of transactions $\mathcal{T}$, denoted by γ, as $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i}$. The primary performance metrics used in the experiments are the scheduling success ratio and the pattern length.

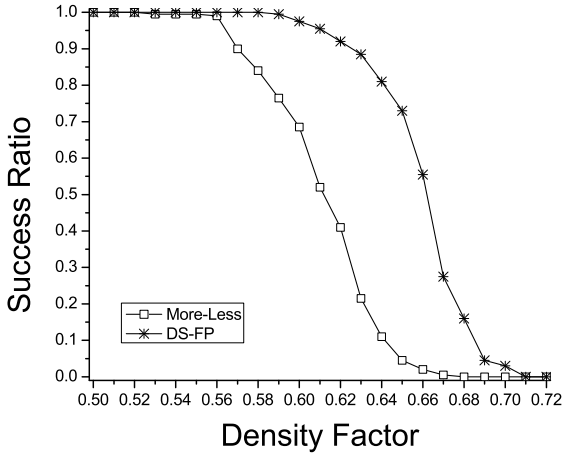

Figure 6.23: Success ratio: *ML* vs. *DS-FP*

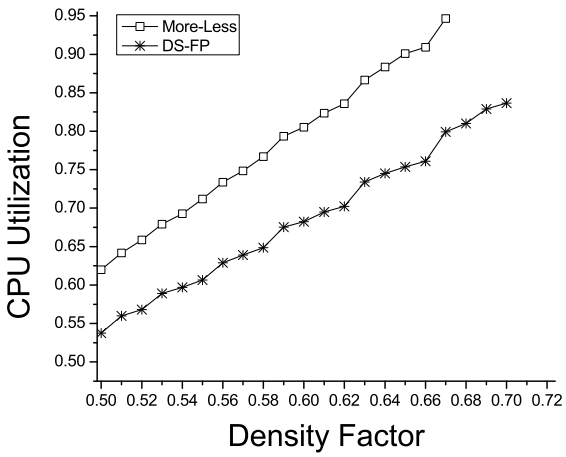

Figure 6.24: CPU workload

Expt. 1: Comparison of Schedulability

In the first set of experiments, we compare the success ratio of *DS-FP* and *ML* in terms of schedulability under various CPU utilizations. In this set of experiments, we run 10 update transactions in the system and vary the *density factor* from 0.50 to 0.72. The increase of the *density factor* is achieved by fixing C_i and decreasing $\mathcal{V}_i$. We conduct 1000 runs for each point and present its average value. Figure 6.23 presents the comparison of the success ratio between *DS-FP* and *ML* under this parameter setting. We observe that *DS-FP* consistently outperforms *ML* in terms of schedulability, which verifies Theorem 6.2.2. As is already implied by Theorem 6.2.2, in the experiment all task sets that are successfully scheduled by *ML* are also schedulable by *DS-FP*. The success ratio of *ML* drops below 0.85 when the *density factor* reaches 0.58. This happens to *DS-FP* only when the *density factor* climbs to 0.64. Also when the *density factor* reaches 0.66, most of the transaction sets cannot be scheduled by *ML* while the success ratio of *DS-FP* is still around 0.55.

The corresponding CPU utilizations of the task sets schedulable under *ML* and *DS-FP* in these experiments are depicted in Figure 6.24, which shows that the CPU utilization

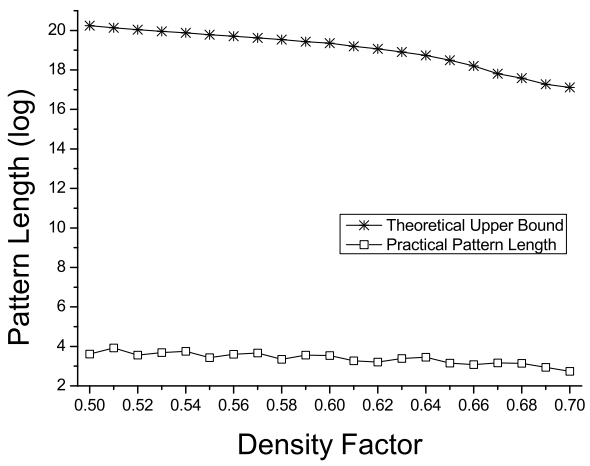

Figure 6.25: Theoretical vs. Practical pattern

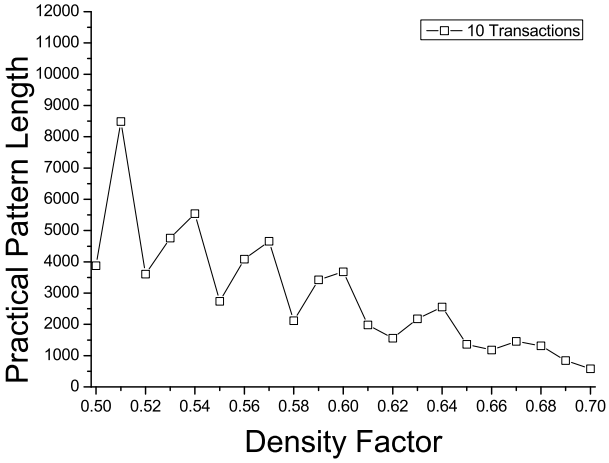

Figure 6.26: Practical pattern length

of *ML* is consistently higher than that of *DS-FP*, and their difference increases when the *density factor* increases. The difference reaches 18% when the *density factor* is 0.67 where the *ML* success ratio approaches 0.

Expt. 2: *DS-FP* Pattern Length

In the second set of experiments, we compare the practical length of the *DS-FP* pattern calculated in Alg. 14 with our theoretical analysis (Corollary. 6.4.2) under the same parameter settings as in Expt. 2.

Figure 6.25 shows the comparison between the theoretical upper bound and the corresponding practical pattern length. We have several observations in Figure 6.25. First, the theoretical upper bound is very large. The y-axis of the figure represents the logarithm of the upper bound and it is exponential w.r.t. the size of the transaction set and $\mathcal{V}_i$. Second, the upper bound decreases while the CPU utilization increases. This is because we fixed the number of update transactions in the system and increased the *density factor* γ. The increase of γ is achieved by decreasing $\mathcal{V}_i$, which decreases the length of the *DS-FP* pattern.

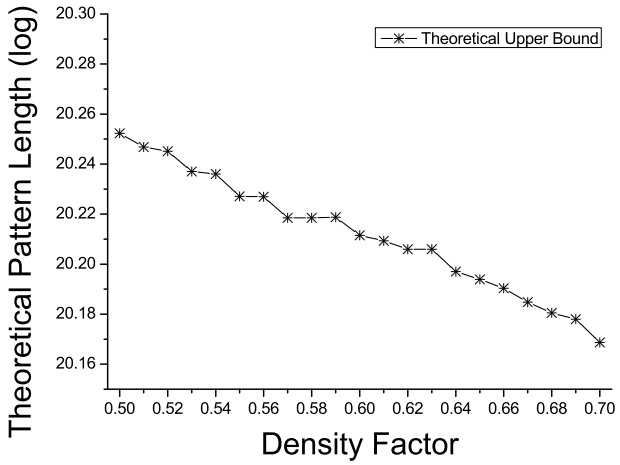

Figure 6.27: Theoretical upper bound

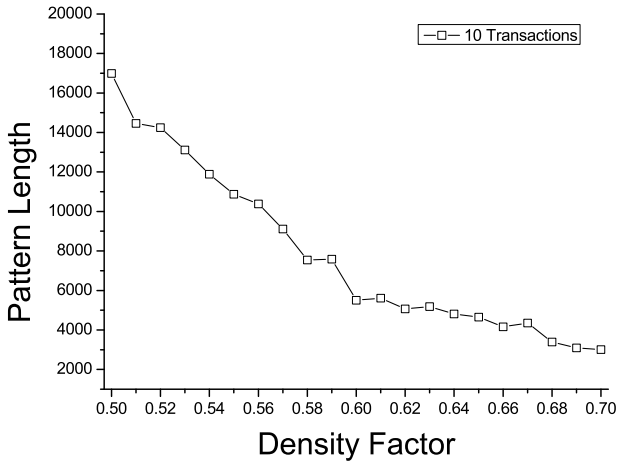

Figure 6.28: Practical pattern length

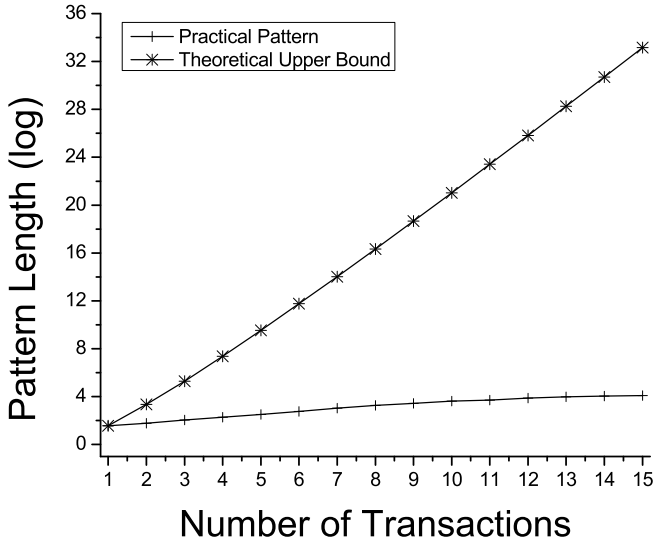

Figure 6.29: Pattern length comparison

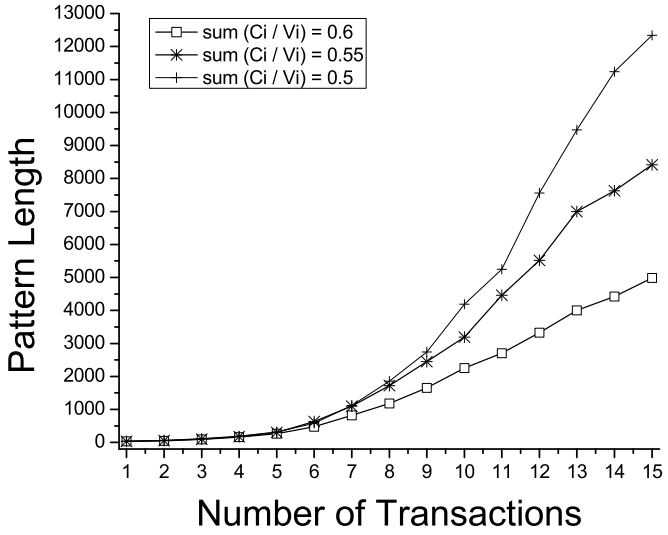

Figure 6.30: Practical pattern length

Compared with the theoretical analysis, Figure 6.25 surprises us by showing a much shorter practical pattern length under the same settings. When the *density factor* reaches 0.69, the ratio between the theoretical upper bound (1.9×10^{17}) and the practical pattern length (8.7×10^{2}), is around 2.18×10^{14}! The reason for this difference lies in the fundamental principle of *DS-FP*. In *DS-FP*, each job $J_{i,j}$ calculates the release time $r_{i,j}$ backwards from its deadline $d_{i,j}$ and this mechanism can easily generate *block*s (where a block is a chunk of continuously occupied time slots) in the *DS-FP* schedule. Given a detected pattern, $\mathcal{P}^{i-1}$, from transaction set $\tau_1, \tau_2, \ldots, \tau_{i-1}$, if there are two jobs of τ_i, $J_{i,j}$ and $J_{i,k}$, whose deadlines have different offsets but lie in the same block in different occurrences of pattern $\mathcal{P}^{i-1}$, their release times should have the same offset in the pattern and a new pattern $\mathcal{P}^{i}$ is detected. In this manner, a pattern that is much shorter than the theoretical analysis can be detected. Similar to the theoretical upper bound, we observe that the average pattern length decreases with the increase of the CPU workload.

We conducted another set of experiments where we fixed the number of update

transactions in the system and increased the *density factor* by increasing C_i. Figure 6.27 and Figure 6.28 summarize our experimental results. In these two figures, we observe that both of the theoretical upper bound and the practical pattern length still decrease as γ increases. This is because in these experiments, the average validity interval is fixed and we increase γ by increasing C_i. According to Theorem 6.4.1, this will decrease the theoretical upper bound on the pattern length. Also, increasing C_i will reduce the number of idle slots in pattern $\mathcal{P}^i$ and according to Eq. 6.45, this further makes it easier to find pattern $\mathcal{P}^{i+1}$ thus finally shorten the pattern length.

Compared with Figure 6.25, we notice that the theoretical pattern length in Figure 6.27 decreases much slower. It only drops 17.5% when the density factor increases from 0.5 to 0.7. In contrast, in Figure 6.25, the pattern length when $\gamma = 0.5$ is 1000 times longer than that when $\gamma = 0.7$. This is because according to our experiment setting where $\mathcal{V}_i$ is much larger than C_i, decreasing $\mathcal{V}_i$ has more significant impact on shortening the pattern length than increasing C_i.

We also evaluate the *DS-FP* pattern length by varying the number of transactions in the system. In our experiments, we fix the *density factor* γ at 0.5, 0.55 and 0.6, respectively with 15 transactions in the system. We generate the transaction set by randomly selecting C_i in $[1, 10]$.

Figure 6.29 compares the logarithm of the practical pattern length and the theoretical upper bound when the *density factor* is fixed at 0.6. Similar to what we observe in Figure 6.25, the upper bound is significantly larger than its corresponding pattern length obtained from our experiments, and it increases when the number of transactions increases.

Figure 6.30 presents the comparison of the actual pattern length under various *density factor* settings. With the same number of transactions, a higher *density factor* setting has lower average $\mathcal{V}_i$ values. The actual pattern length from a higher *density factor* setting is consistently smaller than the one from a lower *density factor* setting.

In summary, it is demonstrated in our experimental studies that *DS-FP* has better

performance compared with *ML* and our schedulability test algorithm can significantly improve the schedulability of the existing one proposed based on *ML* [91]. In addition, the actual pattern length obtained from our experiments is significantly smaller than its corresponding theoretical upper bound. This observation demonstrates that our algorithm is effective for the schedulability test of *DS-FP*.

6.5 Deferrable Scheduling for Dynamic Priority Systems

Since *DS-FP* is a fixed priority scheduling algorithm, this limits its applications for the systems which require to use dynamic priorities for scheduling. To overcome this shortcoming, we propose a dynamic scheduling algorithm, called *Deferrable Scheduling with Least Actual Laxity First (*DS-LALF*)* to maintain the validity of real-time data objects. The *actual laxity* of a job is a measure of the spare time permitted for the job before it misses its deadline – by considering the time needed for higher priority jobs to be executed. It is expected that with the dynamic priority assignment for the update jobs using the *least actual laxity*, a lower update workload can be achieved while the schedulability can be improved. In this section, we first review the *DS-EDF* algorithm in the literature in Section 6.5.1. We then describe the design principle and algorithm details of *DS-LALF* in Section 6.5.2. In Section 6.5.3, based on the pattern analysis technique introduced in [32], we introduce a necessary and sufficient condition for *DS-LALF* and present a schedulability test algorithm to verify the correctness of *DS-LALF* in maintaining the temporal validity of real-time data. We also present a pattern search algorithm to find the shortest and earliest pattern in the *DS-LALF* schedule. We present our performance evaluation on *DS-LALF* in Section 6.5.4.

6.5.1 Deferrable Scheduling with *EDF* (*DS-EDF*)

DS-EDF is the first work in the literature to explore the deferrable scheduling in dynamic priority systems. In *DS-EDF*, the release times of all the first jobs for each update transaction are initialized to be zero and their deadlines are assigned to be the corresponding

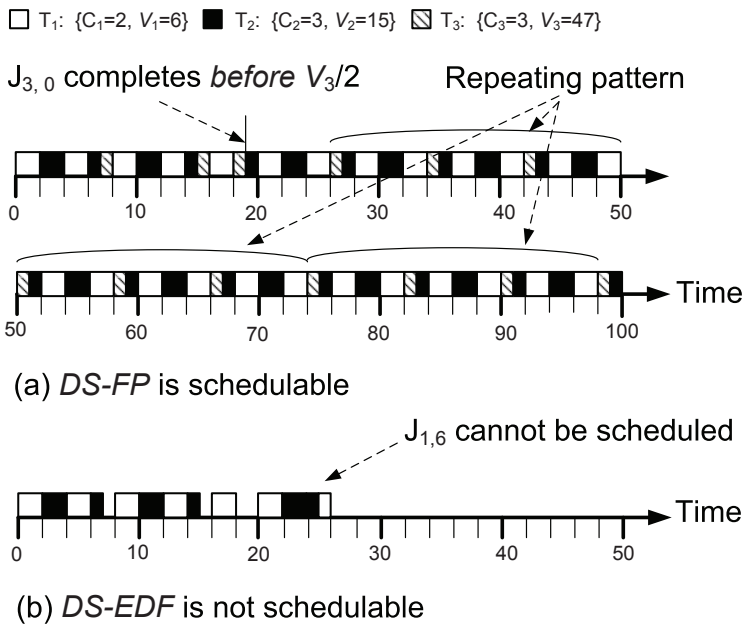

Figure 6.31: *DS-EDF* does not outperform *DS-FP*.

validity intervals. Update jobs to be scheduled are put in a queue Q_{EDF} in the ascending order of their deadlines. Then, the job in Q_{EDF} with the earliest deadline is always scheduled first. For the first jobs of each update transaction, they are scheduled from the designated release time zero, while all other jobs are scheduled *backwards* from their deadlines and derive their release times by considering the total preemption from higher-priority jobs in the schedule. As soon as a job $J_{i,j}$ is completed, its next job $J_{i,j+1}$'s deadline $d_{i,j+1}$, is set to be $r_{i,j+1} + \mathcal{V}_i$, and $J_{i,j+1}$ is enqueued in Q_{EDF}. The algorithm fails when a job is not schedulable, *i.e.*, its calculated release time is smaller than its previous job's deadline.

It is expected that with the use of a more flexible priority assignment than *DS-FP*, the schedulability of *DS-EDF* should be better. Unfortunately, this expectation may not be well placed. The following example shows that there exist scenarios where *DS-FP* can schedule while *DS-EDF* does not.

Example 6.5.1: Consider a set of three update transactions $\{\tau_1, \tau_2, \tau_3\}$ with computation

Job	τ_1	τ_2	τ_3
0	(0,6)	(0, 15)	(0, 47)
1	(4,6)	(10, 15)	
2	(8,10)	(22, 25)	
3	(12,14)		
4	(16,18)		
5	(20,22)		
6	(19,26)		

Table 6.12: Release times and deadlines of update jobs in Example 6.5.1

times 2, 3, 3, and validity intervals 6, 15, 47, respectively. Figure 6.31 (a) depicts a schedule of the transactions using *DS-FP*. The transaction set is schedulable by *DS-FP* because the schedule pattern between time 26 and 50 repeats itself forever. Figure 6.31 (b) depicts a schedule of the transactions using *DS-EDF*. The execution sequence of the update jobs is $J_{1,0}$, $J_{1,1}$, $J_{1,2}$, $J_{1,3}$, $J_{2,0}$, $J_{2,1}$, $J_{1,4}$, $J_{1,5}$, $J_{2,2}$, $J_{1,6}$. The release times and deadlines of the executed jobs are summarized in Table 6.12. Unfortunately, job $J_{1,6}$ cannot be scheduled by *DS-EDF* because its release time, $r_{1,6}$ is 19. It is smaller than its previous job ($J_{1,5}$)'s deadline, which is 22. Notice that by time 26, no job from τ_3 has been scheduled. This is because the first job of τ_3, $J_{3,0}$ has a further deadline, which is 47. □

Example 6.5.1 shows that the pure *EDF* approach may not be a good choice for the deferrable scheduling algorithm. Consider jobs $J_{1,6}$ and $J_{2,2}$, whose deadlines are 26 and 25 respectively. According to *DS-EDF*, $J_{2,2}$ has an earlier deadline and will be first scheduled. However, this will take up all the possible idle time slots to schedule $J_{1,5}$, *i.e.*, time 22 to 24. Instead, if we schedule $J_{1,6}$ first, $r_{2,2}$ will be delayed to time 19. Then, both $J_{1,6}$ and $J_{2,2}$ can be scheduled. Based on this observation, in the next section, we present a new deferrable scheduling algorithm called *Deferrable Scheduling with Least Actual Laxity First* (*DS-LALF*) which can provide a better schedulability than *DS-EDF*.

6.5.2 Deferrable Scheduling with Least Actual Laxity First (*DS-LALF*)

The Principles of *DS-LALF*

In the deferrable scheduling algorithms, to minimize the total update workload while still maintaining data freshness, the release time of each update job is delayed as much as possible and is calculated backwards from its deadline. In this scenario, whether an update job is schedulable or not is decided by whether the derived release time is no smaller than its previous job's deadline by considering its worst-case execution time and the total preemption from higher priority jobs as well. Unfortunately, the pure *EDF* approach adopted by *DS-EDF* fails to take this constraint into consideration, and thus *DS-EDF* is not optimal as illustrated in Example 6.5.1.

Similar to *DS-EDF*, *DS-LALF* is an extension of *DS-FP*. The separation between two consecutive update jobs is determined based on the total preemption from higher priority jobs in the runtime instead of using the worst-case response time. However, different from *DS-EDF* which assigns priority for each update job $J_{i,k}$ according to its deadline, *DS-LALF* decides its priority using the actual laxity, *i.e.*, the number of available idle time slots in the time period $[d_{i,k-1}, d_{i,k}]$ after its previous update job $J_{i,k-1}$ has been completed. The actual laxity takes the aforementioned schedulability constraint on the update jobs into consideration, and it is a better indicator of the urgency of the jobs and is more adaptive to the schedules especially when they are heavily loaded.

The Details of *DS-LALF*

In the followings, we discuss the details of the *DS-LALF* algorithm which is summarized in Alg. 18. In the design of *DS-LALF*, we maintain a queue, Q_{LALF}, for all the update jobs to be scheduled in the system. There is always one job per update transaction available for scheduling in Q_{LALF}. We keep track of D_{max}, the latest deadline among all the jobs that have already been scheduled, and partition all the jobs in Q_{LALF} into two sets S_1 and S_2. The jobs with a deadline no larger than D_{max} are put in S_1 while S_2 contains all the remaining

jobs. *DS-LALF* first schedules the job in S_1 with the least actual laxity. If S_1 is empty, S_2 will be scheduled according to the same rationale. The actual laxity of an update job $J_{a,b}$ is calculated as **CalcIdleSlots**$(d_{a,b-1}, d_{a,b}) - C_a$, where the **CalcIdleSlots**$(d_{a,b-1}, d_{a,b})$ function calculates the number of idle slots in the time interval $[d_{a,b-1}, d_{a,b})$. C_a is the worse-case execution time of $J_{a,b}$.

The release time $r_{i,k}$ of a job $J_{i,k}$ is calculated in **CalcReleaseTime**$(i, k, d_{i,k-1}, d_{i,k})$, where $d_{i,k-1}$ is the deadline of job $J_{i,k-1}$, and $d_{i,k}$ is the deadline of the current job. Once $r_{i,k}$ is computed from Algorithm 19, the deadline of its next job $J_{i,k+1}$ is computed as $d_{i,k+1} = r_{i,k} + \mathcal{V}_i$, and $J_{i,k+1}$ is enqueued in Q_{LALF}. In Algorithm 19, **CalcReleaseTime**(i, k, t_s, t_e) computes the time slots taken by job $J_{i,k}$ in time interval $[t_s, t_e)$, which returns the release time $r_{i,k}$ of $J_{i,k}$, i.e., the earliest time slot that can be taken by $J_{i,k}$ because $r_{i,k}$ is computed backwards from $d_{i,k}$. The worst-case time complexity of **CalcReleaseTime** and **CalcIdleSlots** are both $O(\mathcal{V}_{max})$ where $\mathcal{V}_{max} = \max_{i=1}^{m}(\mathcal{V}_i)$. Also, the worst-case time complexity of enqueue and dequeue can be $O(\ln m)$ if a priority queue is used. Thus, the time complexity of the while loop in *DS-LALF* is $O((m+1) \cdot \mathcal{V}_{max} + 2 \cdot \ln m)$.

DS-LALF can schedule the transaction set in Example 6.5.1 and the generated *DS-LALF* schedule is exactly the same as that by *DS-FP*. However the complexity of *DS-LALF* is much lower than *DS-FP*. Instead of recursively computing the preemptions from all higher priority transactions as in *DS-FP*, *DS-LALF* only needs to calculate the least actual laxities for up to m update jobs in the queue and choose the minimum one to schedule.

6.5.3 *DS-LALF* Schedulability Analysis

In this section, we perform a schedulability analysis for *DS-LALF* based on the pattern analysis technique proposed in [32]. We also provide a schedulability test algorithm for checking the schedulability of *DS-LALF* and a pattern search algorithm for finding the shortest and earliest pattern in a *DS-LALF* schedule.

Alg 18 The Framework of *DS-LALF* Algorithm

Input: A set of update transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ with known $\{C_i\}_{i=1}^{m}$ and $\{\mathcal{V}_i\}_{i=1}^{m}$.
Output: A schedule $\mathcal{S}$ if $\mathcal{T}$ is feasible.

Enqueue all first jobs $J_{i,0}$ of $\tau_i (i = 1, \ldots, m)$ to Q_{LALF} in the ascending order of $\mathcal{V}_i$;
$D_{max} = 0$;
while TRUE **do**
  $S_1 = S_2 = \emptyset$;
  $J_{i,k}$ = NULL; {// The job with least actual laxity}
  $N_{idle} = \mathcal{V}_m$; {// $J_{i,k}$'s actual laxity}
  for each $J_{a,b}$ in Q_{LALF} **do**
    if $d_{a,b} \leq D_{max}$ **then**
      $S_1 = S_1 \bigcup \{J_{a,b}\}$
      if **CalcIdleSlots**$(d_{a,b-1}, d_{a,b}) - C_a < N_{idle}$ **then**
        N_{idle} = **CalcIdleSlots**$(d_{a,b-1}, d_{a,b}) - C_a$;
        $J_{i,k} = J_{a,b}$;
      end if
    else
      if $S_1 == \emptyset$ **then**
        $S_2 = S_2 \bigcup \{J_{a,b}\}$
        if **CalcIdleSlots**$(d_{a,b-1}, d_{a,b}) - C_a < N_{idle}$ **then**
          N_{idle} = **CalcIdleSlots**$(d_{a,b-1}, d_{a,b}) - C_a$;
          $J_{i,k} = J_{a,b}$;
        end if
      else
        break;
      end if
    end if
  end for
  Dequeue $J_{i,k}$ from Q_{LALF};
  $r_{i,k}$ = **CalcReleaseTime**$(i, k, d_{i,k-1}, d_{i,k})$;
  if $r_{i,k} < d_{i,k-1}$ **then**
    return FAILURE;
  end if
  $d_{i,k+1} = r_{i,k} + \mathcal{V}_i$;
  Enqueue $J_{i,k+1}$ to Q_{LALF} in the ascending order of deadlines;
  $D_{max} = \max\{D_{max}, d_{i,k}\}$
end while

Alg 19 CalcReleaseTime(i, k, t_s, t_e)

Input: $J_{i,k}$ and time interval $[t_s, t_e)$.
Output: Release time of $J_{i,k}$, $r_{i,k}$.

```
1: C_R = C_i; {// C_R is the remaining execution time of J_{i,k}.}
2: r_{i,k} = t_e;
3: while r_{i,k} ≥ t_s do
4:    if time slot r_{i,k} is not scheduled then
5:       Schedule time slot r_{i,k} for J_{i,k}
6:       C_R − −;
7:    end if
8:    if C_R == 0 then
9:       return r_{i,k};
10:   end if
11: end while
12: return FAILURE;
```

Pattern Analysis of *DS-LALF*

We denote by a tuple $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$ the *DS-LALF* schedule of length $\mathcal{P}_l$ starting from time $\mathcal{P}_s$. Assume that $J_{\tau,s}$ is the first job of update transaction τ finished no earlier than time t. The state of update transaction τ in *DS-LALF* at time t is denoted by $\mathcal{S}_\tau^Q(t) = (r_\tau^Q(t), f_\tau^Q(t), n_\tau^Q(t))$, where $r_\tau^Q(t)$ and $f_\tau^Q(t)$ are the offsets of $J_{\tau,s}$'s release time and finish time to t respectively. $n_\tau^Q(t)$ is the actual laxity of $J_{\tau,s}$ at time t. Note that $r_\tau^Q(t)$ is positive when $J_{\tau,s}$'s release time is larger than t. Otherwise, $r_\tau^Q(t)$ is zero or negative. $f_\tau^Q(t)$ and $n_\tau^Q(t)$ are always non-negative. We denote $\mathcal{S}_\mathcal{T}^Q(t) = (\mathcal{R}_\mathcal{T}^Q(t), \mathcal{F}_\mathcal{T}^Q(t), \mathcal{N}_\mathcal{T}^Q(t))$ to be the combined $\mathcal{S}_\tau^Q(t)$ states of all the update transactions in $\mathcal{T}$ at time t. Note that once $\mathcal{S}_\mathcal{T}^Q(t)$ is known, the *DS-LALF* schedule from t onward can be determined.

We prove that for each transaction set which is schedulable by *DS-LALF*, there always exists a repeating pattern. First, the following lemma provides an upper bound on the distance between the finish times of any two consecutive jobs in *DS-LALF*.

Lemma 6.5.1: The distance between the finish times of any two consecutive jobs in *DS-LALF* is no larger than $d = \min_i\{\mathcal{V}_i - C_i\}$.

Proof. Assume that $d = \min_i\{\mathcal{V}_i - C_i\} = \mathcal{V}_k - C_k$ and two consecutively finished jobs in *DS-LALF*, $J_{i,p}$ and $J_{j,q}$ have the smallest distance between their finish times which are $f_{i,p}$ and $f_{j,q}$ respectively. We prove the lemma by contradiction. Suppose that $f_{j,q} - f_{i,p} > \mathcal{V}_k - C_k$. Since $J_{i,p}$ and $J_{j,q}$ are consecutively finished jobs, there is no job from τ_k which finishes in the interval $(f_{i,p}, f_{j,q})$. Let $J_{k,s}$ denote the last job of τ_k finished before $f_{i,p}$, and $J_{k,s+1}$ denote the first job of τ_k finished after $f_{j,q}$. According to *DS-LALF*, we have $r_{k,s} \leq f_{k,s} - C_k$ and $f_{k,s+1} - r_{k,s} \geq f_{k,s+1} - f_{k,s} + C_k \geq f_{j,q} - f_{i,p} + C_k > \mathcal{V}_k - C_k + C_k = \mathcal{V}_k$. This violates the validity constraint of τ_k. Therefore, it must not be true that $f_{j,q} - f_{i,p} > \mathcal{V}_k - C_k$, *i.e.*, the distance between the finish times of any two consecutive jobs in *DS-LALF* is no larger than $d = \min_i\{\mathcal{V}_i - C_i\}$. □

The following theorem states that a *DS-LALF* schedule has a repeating pattern that must occur at least once in a bounded time interval.

Theorem 6.5.1: Given an update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ $(1 \leq i \leq m)$, if it can be scheduled by *DS-LALF*, then the *DS-LALF* schedule has a fixed repeating pattern that occurs at least once in the interval

$$[\mathcal{V}_m, \mathcal{V}_m + d \cdot (\prod_{i=1}^{m} (2(\mathcal{V}_i - C_i) + 1) \cdot (\mathcal{V}_i - C_i + 2)^2)]$$

Proof. In *DS-LALF*, there is always one job per update transaction available for scheduling in Q_{LALF}. The theorem can be proven by the following two claims.

- In the time interval $[\mathcal{V}_m, \mathcal{V}_m + d \cdot (\prod_{i=1}^{m} (2(\mathcal{V}_i - C_i) + 1) \cdot (\mathcal{V}_i - C_i + 2)^2)]$, there exist two time points, t_1 and t_2, so that t_1 and t_2 are two jobs' finish times (not necessarily from the same transaction), and $\mathcal{S}_{\mathcal{T}}^{Q}(t_1) = \mathcal{S}_{\mathcal{T}}^{Q}(t_2)$.
- The schedule in the interval $[t_1, t_2]$ is a fixed pattern repeating itself thereafter.

The proof of the first claim relies on the *Pigeonhole* principle [30]. At time t_1, it is assumed that the job of update transaction τ_i in Q_{LALF} is $J_{i,j}$ and its state is $\mathcal{S}_{\tau_i}^{Q}(t_1) =$

$(r^Q_{\tau_i}(t_1), f^Q_{\tau_i}(t_1), n^Q_{\tau}(t_1))$. Since $J_{i,j}$ is the first job of τ_i to be finished no earlier than t_1 and $J_{i,j-1}$ is finished before t_1, $f^Q_{\tau_i}(t_1)$ can only be one of $(\mathcal{V}_i - C_i + 1)$ possible values (pigeonholes) in $[0, \mathcal{V}_i - C_i]$. Similarly, $r^Q_{\tau_i}(t_1)$ can only be one of $2 \cdot (\mathcal{V}_i - C_i)$ possible values (pigeonholes) in $[-\mathcal{V}_i + C_i, \mathcal{V}_i - C_i)$ and the possible values (pigeonholes) of $n^Q_{\tau_i}(t_1)$ is $(\mathcal{V}_i - C_i + 1)$ from 0 to $\mathcal{V}_i - C_i$. As $\mathcal{V}_m$ is the finish time of $J_{m,1}$, let us take $\mathcal{V}_m$ as t_1 and look at the finish times of $\prod_{i=1}^{m} (2 \cdot (\mathcal{V}_i - C_i) + 1) \cdot (\mathcal{V}_i - C_i + 2)^2)$ consecutively executed jobs from t_1. It follows from the *Pigeonhole* principle that there must exist a job's finish time t_2 such that $\mathcal{S}^Q_{\mathcal{T}}(t_1) = \mathcal{S}^Q_{\mathcal{T}}(t_2)$. According to Lemma 6.5.1, the distance between t_1 and t_2 is no larger than $d \cdot (\prod_{i=1}^{m} (2 \cdot (\mathcal{V}_i - C_i) + 1) \cdot (\mathcal{V}_i - C_i + 2)^2)$.

To prove the second claim, we assume that the sequence of executed jobs from t_1 and t_2 are $\mathcal{S}_1 = \{J^0_1, J^1_1, J^2_1, \ldots\}$ and $\mathcal{S}_2 = \{J^0_2, J^1_2, J^2_2, \ldots\}$, respectively, and the corresponding job finish times are $\mathcal{F}_1 = \{t^0_1, t^1_1, t^2_1, \ldots\}$ and $\mathcal{F}_2 = \{t^0_2, t^1_2, t^2_2, \ldots\}$, respectively. We also assume that the actual laxities of the corresponding jobs are $\mathcal{N}_1 = \{n^0_1, n^1_1, n^2_1, \ldots\}$ and $\mathcal{N}_2 = \{n^0_2, n^1_2, n^2_2, \ldots\}$, respectively. We will prove that at any time t^k_1 and t^k_2 ($k \geq 0$), we have $\mathcal{F}^Q_{\mathcal{T}}(t^k_1) = \mathcal{F}^Q_{\mathcal{T}}(t^k_2)$ and $\mathcal{N}^Q_{\mathcal{T}}(t^k_1) = \mathcal{N}^Q_{\mathcal{T}}(t^k_2)$, and the schedules in $[t_1, t^k_1]$ and $[t_2, t^k_2]$ are the same after the execution of J^k_1 and J^k_2.

Proof for case $k = 0$ is obvious because $t_1 = t^0_1$ and $t_2 = t^0_2$. $\mathcal{S}^Q_{\tau_i}(t^0_1) = \mathcal{S}^Q_{\tau_i}(t^0_2)$ implies $(\mathcal{F}^Q_{\mathcal{T}}(t^0_1) = \mathcal{F}^Q_{\mathcal{T}}(t^0_2)) \wedge (\mathcal{N}^Q_{\mathcal{T}}(t^0_1) = \mathcal{N}^Q_{\mathcal{T}}(t^0_2))$ and the schedule in the time intervals $[t_1, t^0_1]$ and $[t_2, t^0_2]$ are both empty.

Assume that the statement is true at time t^i_1 and t^i_2, then because $\mathcal{F}^Q_{\mathcal{T}}(t^i_1) = \mathcal{F}^Q_{\mathcal{T}}(t^i_2)$, it is obvious that J^{i+1}_1 and J^{i+1}_2 are from the same update transaction and their deadlines t^{i+1}_1 and t^{i+1}_2 have the same offset to t^i_1 and t^i_2 respectively. Let us assume that $J^{i+1}_1 = J_{k,p}$ and $J^{i+1}_2 = J_{k,q}$. We will prove that the statement is also true at time t^{i+1}_1 and t^{i+1}_2. We will discuss the following two cases:

Case I: $J_{k,p}$ and $J_{k,q}$ are in $Q_{LALF}(t_1)$ and $Q_{LALF}(t_2)$. In this case, we have $\mathcal{S}^Q_{\tau_k}(t_1) = \mathcal{S}^Q_{\tau_k}(t_2)$. Thus, the offsets of their release times to t_1 and t_2 are already known and are equal to each other. If there is no item in $\mathcal{F}_1$ and $\mathcal{F}_2$ before t^{i+1}_1 and t^{i+1}_2 respectively that has a larger value

of deadline, then all time slots in the intervals $[t_1^i, t_1^{i+1}]$ and $[t_2^i, t_2^{i+1}]$ are idle. Otherwise, assume that t_1^h and $t_2^h (h \leq i)$ in $\mathcal{F}_1$ and $\mathcal{F}_2$ have the largest values satisfying $t_1^h > t_1^{i+1}$ and $t_2^h > t_2^{i+1}$, then because we have already proven that the schedules in $[t_1, t_1^h]$ and $[t_2, t_2^h]$ are the same, the schedules in $[t_1, t_1^{i+1}]$ and $[t_2, t_2^{i+1}]$ are also the same before the execution of J_1^{i+1} and J_2^{i+1}. No matter in which case, the schedule in $[t_1, t_1^{i+1}]$ and $[t_2, t_2^{i+1}]$ will still be the same after the execution of J_1^{i+1} and J_2^{i+1}. Also, according to *DS-LALF*, we have $f_{k,p+1} = r_{k,p} + \mathcal{V}_k$ and $f_{k,q+1} = r_{k,q} + \mathcal{V}_k$, and they have the same offset to t_1^{i+1} and t_2^{i+1} respectively. Thus at t_1^{i+1} and t_2^{i+1}, we have $\mathcal{F}_{\mathcal{T}}^{Q}(t_1^{i+1}) = \mathcal{F}_{\mathcal{T}}^{Q}(t_2^{i+1})$. Similar to the analysis above, we can also derive that the schedules in $[t_1^{i+1}, f_{k,p+1}]$ and $[t_2^{i+1}, f_{k,q+1}]$ are the same and we have $\mathcal{N}_{\mathcal{T}}^{Q}(t_1^{i+1}) = \mathcal{N}_{\mathcal{T}}^{Q}(t_2^{i+1})$.

Case II: $J_{k,p}$ and $J_{k,q}$ are not in $Q_{LALF}(t_1)$ and $Q_{LALF}(t_2)$. In this case, we need to calculate their release time backwards from t_1^{i+1} and t_2^{i+1} according to *DS-LALF*. Since $J_{k,p}$ and $J_{k,q}$ are not the first job of τ_k finished after t_1 and t_2, their previous jobs $J_{k,p-1}$ and $J_{k,q-1}$ must be finished after t_1 and t_2 and no later than J_1^i and J_2^i in the execution sequence in $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively. According to the schedulability constraint in *DS-LALF*, we have $f_{k,p-1} \leq r_{k,p} \leq t_1^{i+1}$ and $f_{k,q-1} \leq r_{k,q} \leq t_2^{i+1}$. Similar to the analysis in Case I, we have that the schedules in $[t_1, t_1^{i+1}]$ and $[t_2, t_2^{i+1}]$ are the same before the execution of $J_{k,p}$ and $J_{k,q}$. Thus, the calculated $r_{k,p}$ and $r_{k,q}$ will have the same offset to t_1^{i+1} and t_2^{i+1}, and the finish times of their next jobs $J_{k,p+1}$ and $J_{k,q+1}$ will have the same offset to t_1^{i+1} and t_2^{i+1}, *i.e.*, $\mathcal{F}_{\mathcal{T}}^{Q}(t_1^{i+1}) = \mathcal{F}_{\mathcal{T}}^{Q}(t_2^{i+1})$. With the similar analysis in Case I, we will also have $\mathcal{N}_{\mathcal{T}}^{Q}(t_1^{i+1}) = \mathcal{N}_{\mathcal{T}}^{Q}(t_2^{i+1})$, and the schedules in $[t_1, t_1^{i+1}]$ and $[t_2, t_2^{i+1}]$ are the same after the execution of J_1^{i+1} and J_2^{i+1}.

This induction continues until we reach t_2, and we have that the schedules in $[t_1, t_2]$ and $[t_2, 2 \cdot t_2 - t_1]$ are the same. This finishes the proof. □

According to the theorem, if an update transaction set can be scheduled by *DS-LALF* in the interval $[\mathcal{V}_m, \mathcal{V}_m + d \cdot (\prod_{i=1}^{m} (2(\mathcal{V}_i - C_i) + 1) \cdot (\mathcal{V}_i - C_i + 2)^2)]$, then it is schedulable by *DS-LALF* because a fixed pattern appearing in the interval repeats itself forever. Thus we have the following corollary.

Corollary 6.5.1: An update transaction set $\mathcal{T}$ can be scheduled by *DS-LALF* if and only if in the interval $[\mathcal{V}_m, \mathcal{V}_m + d \cdot (\prod_{i=1}^{m} (2(\mathcal{V}_i - C_i) + 1) \cdot (\mathcal{V}_i - C_i + 2)^2)]$, it can be scheduled by *DS-LALF*.

DS-LALF Pattern Search Algorithm

Theorem 6.5.1 proves the existence of a repeating pattern for a given *DS-LALF* schedule. Given a repeating pattern $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$, the following corollary follows directly from the fact that all update transactions have the same states at times $\mathcal{P}_s + t$ and $\mathcal{P}_s + t + \mathcal{P}_l$ for $t > 0$, i.e., $\mathcal{S}_{\mathcal{T}}^{\mathcal{Q}}(\mathcal{P}_s + t) = \mathcal{S}_{\mathcal{T}}^{\mathcal{Q}}(\mathcal{P}_s + t + \mathcal{P}_l)$.

Corollary 6.5.2: If $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$ is a pattern repeating itself from time $\mathcal{P}_s$, then $(\mathcal{P}_s + t, \mathcal{P}_l)$ $(t > 0)$ is also a pattern repeating itself from time $\mathcal{P}_s + t$.

Now, we present the pattern search algorithm to find the shortest and earliest pattern in the schedule. The algorithm is summarized in Alg. 20. It follows the idea in Theorem 6.5.1 that if the transaction set $\mathcal{T}$ has the same state at two time points which are the deadlines of two different jobs, the schedule in the time interval between them forms a pattern and will repeat it forever. Starting from time $\mathcal{V}_m$, we compare the state of the transaction set $\mathcal{T}$ at each time point t which is a certain job's deadline backward to those time points in $[\mathcal{V}_m, t)$ which are also deadlines of certain jobs. Theorem 6.5.1 guarantees that either we can find a repeating pattern by checking at most $\prod_{i=1}^{m} (2 \cdot (\mathcal{V}_i - C_i) + 1) \cdot (\mathcal{V}_i - C_i + 2)^2)$ time points or the transaction set $\mathcal{T}$ is unschedulable by *DS-LALF*.

Theorem 6.5.2: $\mathcal{P}$ returned by the pattern search algorithm is the shortest and earliest pattern.

Proof: We prove the theorem by contradiction. Suppose the pattern returned by Alg. 20 is $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$ and there exists another pattern $\mathcal{P}' = (\mathcal{P}'_s, \mathcal{P}'_l)$ with $\mathcal{P}'_l < \mathcal{P}_l$. Assuming $\mathcal{P}_s + (k-1) \cdot \mathcal{P}_l < \mathcal{P}'_s \leq \mathcal{P}_s + k \cdot \mathcal{P}_l (k \geq 1)$, following Corollary 6.5.2 we have $\mathcal{S}_{\mathcal{T}}^{\mathcal{Q}}(\mathcal{P}_s + k \cdot \mathcal{P}_l) =$

Alg 20 Pattern Searching Algorithm for *DS-LALF*

Input: A successful *DS-LALF* schedule $\mathcal{S}$.
Output: The earliest and shortest *DS-LALF* pattern $\mathcal{P}$.

Sort the job deadlines at and after $\mathcal{V}_m$ in $\mathcal{S}$ in ascending order, $\{t_0, t_1, t_2, \ldots, t_k, \ldots\}$ where $t_0 = \mathcal{V}_m$
$N_{max} \leftarrow 1 + \prod_{i=1}^{m} (2 \cdot (\mathcal{V}_i - C_i) + 1) \cdot (2 \cdot (\mathcal{V}_i - C_i + 1) + 1)$
$k \leftarrow 0$;

while $k \leq N_{max}$ **do**
  $k \leftarrow k + 1$;
  for $h = k - 1$ to 1 **do**
    {// Assume that t_k is the deadline of job $d_{i,j}$}
    if $t_h \leq d_{i,j-1}$ and $\mathcal{S}^Q_{\mathcal{T}}(t_h) == \mathcal{S}^Q_{\mathcal{T}}(t_k)$ **then**
      $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l) \leftarrow [t_h, t_k)$
      while $\mathcal{S}^Q_{\mathcal{T}}(\mathcal{P}_s - 1) = \mathcal{S}^Q_{\mathcal{T}}(\mathcal{P}_s - 1 + \mathcal{P}_l)$ **do**
        $\mathcal{P} \leftarrow (\mathcal{P}_s - 1, \mathcal{P}_l)$
      end while
      return $\mathcal{P}$;
    else
      continue;
    end if
  end for
end while
return **No pattern found**;

Alg 21 Schedulability Test Algorithm for *DS-LALF*

Input: A transaction set $\mathcal{T}$.

Output: Whether $\mathcal{T}$ is schedulable under *DS-LALF*.

Sort the job deadlines at and after $\mathcal{V}_m$ in $\mathcal{S}$ in ascending order, $\{t_0, t_1, t_2, \ldots, t_k, \ldots\}$ where $t_0 = \mathcal{V}_m$
$N_{max} \leftarrow 1 + \prod_{i=1}^{m} (2 \cdot (\mathcal{V}_i - C_i) + 1) \cdot (2 \cdot (\mathcal{V}_i - C_i + 1) + 1)$
$N_{ex} \leftarrow \sum_i \lceil \frac{d_{max}}{\mathcal{V}_i - C_i} \rceil$
$I_c \leftarrow 0$; $k \leftarrow 0$;

while $I_c \leq N_{max}$ **do**
  $I_o \leftarrow I_c$;
  if $k < N_{max} + N_{ex}$ **then**
    $k \leftarrow k + 1$;
    {// Assume that t_k is the deadline of $\tau_{i,j}$ and t_c is the earliest time point larger than $t_k - (\mathcal{V}_i - C_i)$}
    Schedule $\tau_{i,j}$ and derive its release time $r_{i,j}$
    if $r_{i,j} < d_{i,j-1}$ **then**
      return FALSE;
    else
      for $h = c$ to k **do**
        Adding $r_{i,j}$ into $\mathcal{S}^{Q}_{\mathcal{T}}(t_h)$
        if $\mathcal{S}^{Q}_{\mathcal{T}}(t_h)$ is complete **then**
          $I_c \leftarrow h$;
        end if
      end for
    end if
    for $r = I_o$ to I_c **do**
      {// Assume that t_r is the deadline of $\tau_{p,q}$}
      for $s = r - 1$ to 0 **do**
        if $t_s \leq d_{p,q-1}$ and $\mathcal{S}^{Q}_{\mathcal{T}}(t_s) = \mathcal{S}^{Q}_{\mathcal{T}}(t_r)$ **then**
          $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l) \leftarrow [t_s, t_r)$
          return TRUE;
        end if
      end for
    end for
  end if
end while

$\mathcal{S}^{Q}_{\mathcal{T}}(\mathcal{P}_s + k \cdot \mathcal{P}_l + \mathcal{P}'_l)$. This further derives $\mathcal{S}^{Q}_{\mathcal{T}}(\mathcal{P}_s) = \mathcal{S}^{Q}_{\mathcal{T}}(\mathcal{P}_s + \mathcal{P}'_l)$ and thus $[\mathcal{P}_s, \mathcal{P}_s + \mathcal{P}'_l)$ is also a pattern. However, this pattern should be found in Line 12 in Alg. 20 before pattern $\mathcal{P}$ is found. So $\mathcal{P}$ must be the shortest pattern. Similarly, suppose there exists another pattern $\mathcal{P}'' = (\mathcal{P}''_s, \mathcal{P}''_l)$ with $\mathcal{P}''_l \geq \mathcal{P}_l$ and $\mathcal{P}''_s + \mathcal{P}''_l < \mathcal{P}_s + \mathcal{P}_l$. We can derive $[\mathcal{P}''_s, \mathcal{P}''_s + \mathcal{P}_l)$ is also a pattern which should be found before pattern $\mathcal{P}''$. So $\mathcal{P}$ must also be the earliest pattern. This finishes the proof. □

DS-LALF Schedulability Test Algorithm

The schedulability test for *DS-LALF* is more complicated than its pattern search algorithm. The input of the pattern search algorithm (Alg. 20) is a successful schedule and at each time point t_k, the state of the transaction set $\mathcal{T}$ is available for comparison. However, when we test the schedulability of $\mathcal{T}$ at run time, the state information may not be complete. This is because at the job execution time, the release times of the jobs in its Q_{LALF} may have not been derived yet. For this reason, the schedulability test at time $t_k (k \geq 1)$ has to be delayed until all the state information at time $t_i (1 \leq i \leq k)$ is available. To make sure that the job that finishes at $t_{N_{max}}$ has complete state information, we need to execute more jobs after time $t_{N_{max}}$. Let us use N_{ex} to denote this number and $d_{max} = \max_i\{\mathcal{V}_i - C_i\}$, we have $N_{ex} \leq \sum_i \lceil \frac{d_{max}}{\mathcal{V}_i - C_i} \rceil$.

Alg. 21 presents the framework of the *DS-LALF* schedulability test algorithm. The algorithm records the state information of the transaction set $\mathcal{T}$ at the finish time of each executed job and keeps an index k for the current job to be executed. It also maintains an index I_c to record the latest state whose information is complete. In Alg. 21, every time it schedules a job, it derives its release time backwards from its deadline (Line 11). If the job is not schedulable, the transaction set $\mathcal{T}$ does not pass the schedulability test and a failure will be reported (Line 13). Otherwise, it will install the derived release time to previous states whose information is incomplete and update the index I_c accordingly (Line 15 to Line 20). If a new state with complete information is identified (Line 17), it will be compared with

previous states for pattern searching (Line 22 to Line 30). This process continues until either a failure is reported or a pattern is found.

Remarks: Please note that in the schedulability analysis presented above, we use the existence of an exact repeating pattern to test the schedulability of the update transaction set $\mathcal{T}$ under *DS-LALF*. However, if we only need to verify $\mathcal{T}$'s schedulability and identifying the exact repeating pattern in the constructed *DS-LALF* schedule is not necessary, then the efficiency of the schedulability test algorithm can be further improved. In the improved algorithm, the requirement in Theorem 6.5.1 that the actual laxity for each update transaction τ at time point t_1 is equal to that at time point t_2, *i.e.*, $\forall \tau \in \mathcal{T}, n_\tau^Q(t_1) = n_\tau^Q(t_2)$ can be relaxed to be $\forall \tau \in \mathcal{T}, n_\tau^Q(t_1) \leq n_\tau^Q(t_2)$. With this relaxation, if the remaining execution time for each update transaction τ at time t_2 is no less than that at t_1, then as long as the transaction set $\mathcal{T}$ in time interval $[t_1, t_2)$ is schedulable, we can repeat the schedule in $[t_1, t_2)$ from time point t_2 by assigning $n_\tau^Q(t_1)$ to $n_\tau^Q(t_2)$ for each update transaction τ. In such cases, the processor may idle after the completion of τ's job until the job's assigned time slots expire.

6.5.4 Performance Evaluation of *DS-LALF*

In this set of experiments, we evaluate the performance of the *DS-LALF* algorithm in maintaining real-time data validity. The parameter settings in the experiments follow the one used in Section 6.4.7 and are summarized in Table 6.10 and Table 6.11. The primary performance metrics used in the experiments are the CPU utilization imposed by the update transactions, the success ratio of schedulability, the theoretical bounds and practical lengths of the schedule patterns constructed by different scheduling algorithms.

Comparison of CPU Utilization

In the first set of experiments, we quantitatively compare the CPU utilization of scheduling the same set of update transactions using *DS-FP*, *DS-LALF* and *ML* in both the worst-case and general case scenarios. For the ease of comparison, we assume that for each

update transaction τ_i, its worst-case execution time C_i is uniformly selected in $[1, 5]$ and its validity interval $\mathcal{V}_i$ is uniformly distributed in $[40, 200]$. In the worst-case scenario, $c_{i,j}$, the execution time of $J_{i,j}$, is always set to C_i. In the general case, $c_{i,j}$ is chosen using a uniform distribution between 1 and C_i.

As shown in Figure 6.32, the CPU utilization of both *DS-FP* and *DS-LALF* is consistently lower than that of *ML* in both the worse-case and general case scenarios especially when the number of update transactions is larger. The improvement over *ML* reaches 16.8% for *DS-FP* and 19.4% for *DS-LALF* in the worse-case, and 17.5% for *DS-FP* and 22.8% for *DS-LALF* in the general case when there are 24 update transactions in the system. In addition, consistent with our expectation, the CPU utilization of the three methods in the general case is around half of that in the worst-case. This is because, in the general case, although *ML*, *DS-FP* and *DS-LALF* still assign the worst-case execution time to a job $J_{i,j}$ to keep the pattern, $J_{i,j}$'s actual execution time, $c_{i,j}$, follows the uniform distribution between 1 and C_i. Thus, the CPU will be idle after completing a job until its assigned time slots expire. Another important observation from Figure 6.32 is that the CPU utilization of *DS-LALF* is consistently slightly lower than that of *DS-FP* and significantly lower than that of *ML* especially in the general case and with larger number of update transactions. This indicates that the performance of *DS-LALF* is better than *DS-FP* as well as *ML* in terms of CPU utilization as it adaptively schedules the jobs based on their urgencies according to the number of available time slots to schedule the pending update jobs.

Comparison of Schedulability

In the second set of experiments, we compare the success ratio of schedulability among *DS-LALF*, *DS-EDF*, *DS-FP* and *ML* under various update workloads by changing the density factor. Figure 6.33 depicts the success ratio of schedulability of *DS-LALF*, *DS-EDF*, *DS-FP* and *ML* when the density factor is varied from 0.35 to 0.65 for a system consisting of 5 update transactions with execution time always set to the worst-case value. The increase

in density factor is achieved by fixing C_i and decreasing value for $\mathcal{V}_i$. We have conducted 5,000 runs for each setting and present the average values from them in the figure.

As shown in Figure 6.33, *DS-LALF* consistently outperforms *ML* and *DS-EDF* in terms of success ratio of schedulability but slightly lower than that of *DS-FP*. The success ratios of *ML* and *DS-EDF* drop below 0.85 when the density factor is 0.55. This happens to *DS-FP* and *DS-LALF* only when the density factors are 0.6 and 0.59 respectively. Also, when the density factor is 0.63, almost all the transaction sets cannot be scheduled by *ML* and *DS-EDF* while the success ratios of *DS-FP* and *DS-LALF* are still around 0.27 and 0.11 respectively.

Another observation from Figure 6.33 is that the success ratio of schedulability of *DS-EDF* is quite varied for different values of density factor. For low density factor values, its schedulability is similar to *ML*, *DS-LALF* and *DS-FP* as the workload is light. However, when the density factor is more than 0.45, its schedulability varies a lot and is even lower than that of *ML*. On the contrary, when the density factor is more than 0.56, its schedulability becomes significantly higher than that of *ML* but is still lower than *DS-LALF* and *DS-FP*. The main reason for the poor and unstable schedulability of *DS-EDF* as compared with *DS-LALF* and *DS-FP* is that introducing a dynamic scheduling mechanism to schedule update jobs using earliest deadline first may make some jobs miss the deadlines after being served by the CPU for a long time. This is consistent with the general performance of the earliest deadline first scheduling [54].

Comparison of Pattern Lengths

In the final set of experiments, we compare the pattern lengths (called practical pattern lengths) of *DS-FP*, *DS-EDF* and *DS-LALF* obtained from the corresponding pattern searching algorithms and the pattern lengths (called the theoretical upper bound) obtained from the theoretical analysis. The parameter settings of the experiments are the same as those in previous experiments.

Figure 6.34 compares the theoretical upper bounds and the corresponding practical pattern lengths of *DS-FP*, *DS-EDF* and *DS-LALF*. As shown in the figure, the theoretical upper bounds are very large compared with the corresponding practical pattern lengths. When the density factor is 0.5, the ratio between the theoretical upper bound (1.01×10^7) and the practical pattern length (2.4×10^2) of *DS-FP* is about 5×10^4. The ratios for *DS-EDF* and *DS-LALF* are even larger as can be calculated from the results. The reason for the great difference lies in the fundamental principles of *DS-FP*, *DS-EDF* and *DS-LALF*. In all these three algorithms, each update job $J_{i,j}$ calculates the release time $r_{i,j}$ backwards from its deadline $d_{i,j}$. This mechanism can easily generate *block*s. Each block is a chunk of continuously occupied time slots in a *DS-FP*, *DS-EDF* or *DS-LALF* schedule. For instance, in *DS-FP*, given a detected pattern, $\mathcal{P}^{i-1}$, from transaction set $\tau_1, \tau_2, \ldots, \tau_{i-1}$, if there are two jobs of τ_i, $J_{i,j}$ and $J_{i,k}$, whose deadlines have different offsets but lie in the same block in different occurrences of pattern $\mathcal{P}^{i-1}$, their release times should have the same offset in the pattern and a new pattern $\mathcal{P}^i$ is detected. In this manner, a pattern that is much shorter than the theoretical analysis can be detected.

Similar to the theoretical upper bound shown in Figure 6.34, we observe that the practical pattern lengths of *DS-FP*, *DS-EDF* and *DS-LALF* decrease gradually with an increase in density factor as shown in Figure 6.35. This is because an increase in density factor is achieved by decreasing $\mathcal{V}_i$, which decreases the pattern lengths for all the three algorithms. In addition, the pattern length of *DS-LALF* is much smaller than that of *DS-EDF* but slightly larger than that of *DS-FP*. As shown in Figure 6.35, the practical pattern length of *DS-EDF* varies a lot with an increase in density factor although the general trend is decreasing. This is because the earliest deadline first scheduling in *DS-EDF* introduces addition variations into the patterns making them have longer and less stable pattern.

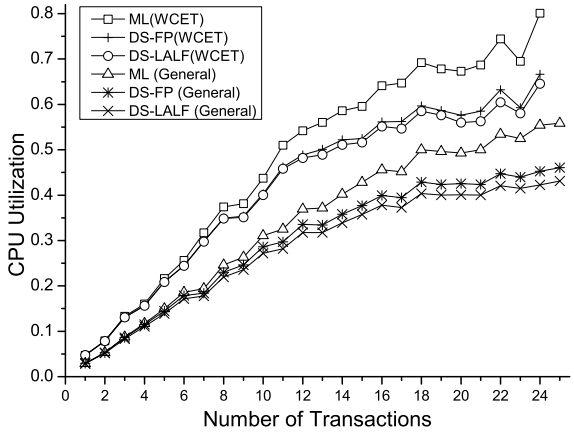

Figure 6.32: CPU utilization

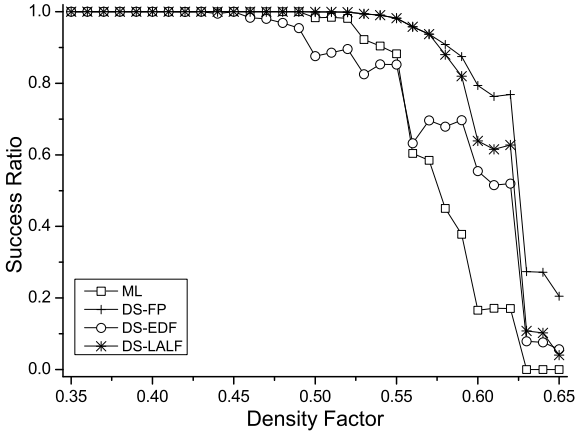

Figure 6.33: Success ratio of schedulability

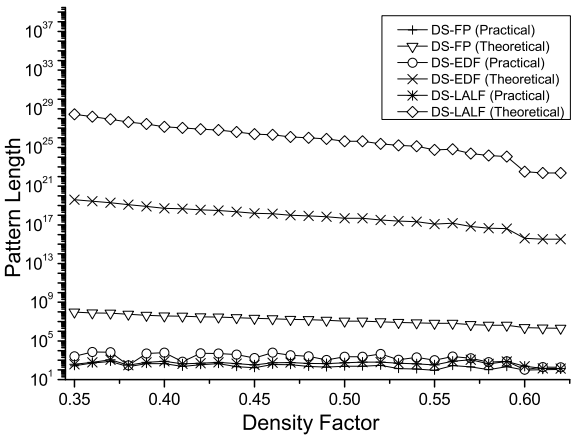

Figure 6.34: Pattern length comparison

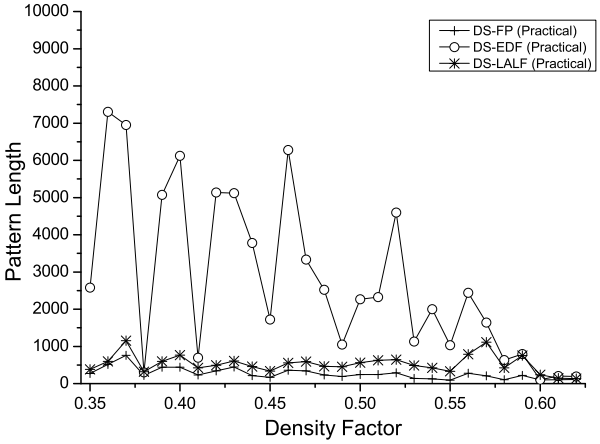

Figure 6.35: Practical pattern length

6.6 Summary

This chapter proposed two novel deferrable scheduling algorithms for maintaining data freshness in large-scale cyber-physical systems. The first algorithm, *DS-FP*, is designed for fixed priority transactions. Distinct from past studies in which the periodic task model is adopted, *DS-FP* adopts the *sporadic* task model. The deadlines of jobs and separation of two consecutive jobs of an update transaction are adjusted judiciously so that the farthest distance of the sampling time of a job and the completion time of its next job is bounded by the validity length of the updated real-time data. We presented a necessary and sufficient condition for its schedulability based on pattern analysis. We proposed a theoretical estimation of the processor utilization of *DS-FP*, which is verified in our experimental studies. Moreover, we also presented novel and practical approaches derived from *DS-FP*, namely *DESH-SC* and *DESH-SA*, that reduce on-line scheduling overhead to $O(1)$. It is also demonstrated in our experiments that *DS-FP* greatly reduces processor workload compared to *ML*. Thus, *DS-FP* can improve the performance of application transactions when it is used by a RTDBS to track environmental changes. Our experiments demonstrate that *DESH-SA* is a very effective approach for minimizing sensor update workload while guaranteeing the validity constraint, and it is efficient in terms of time and space complexity.

To further reduce the online computational overhead of *DS-FP* and make it applicable for systems requiring dynamic priority assignment, we enhanced *DS-FP* and proposed a dynamic scheduling algorithm, called *Deferrable Scheduling with Least Actual Laxity First (*DS-LALF*)* to maintain the validity of real-time data objects. *DS-LALF* applies the same principle of deferrable scheduling as *DS-FP* to defer the release times of update jobs as late as possible, so that the separation time between two consecutive update jobs from the same update transaction can be maximized and the CPU workload incurred by the update transactions can be minimized. Instead of assigning the priorities of the update transactions according to Shortest Validity First policy (*SVF*), *DS-LALF* assigns the priorities of the update jobs according to their actual laxities in the run time. The actual laxity of a job is a

measure of the spare time the job has before it misses its deadline – by considering the time needed for higher priority jobs to be executed. Similar to *DS-FP*, we present a necessary and sufficient condition for the feasibility of *DS-LALF*, along with a pattern search algorithm to find the shortest and earliest pattern in the *DS-LALF* schedule. Our experimental results show that *DS-LALF* has a much lower online computational overhead and incurs lower update workload compared with *DS-FP*. Its schedulability is close to *DS-FP* but is much better than the approaches under periodic task model, such as *HH* and *ML* and other dynamic scheduling algorithms in the literature, like *DS-EDF* for instance.

Chapter 7

Maintaining Data Freshness in Dynamic Cyber-Physical Systems

Most research efforts in the literature on maintaining real-time data freshness including *DS-FP* and *DS-LALF* presented in Chapter 6 assume that the set of real-time data objects and their quality requirements in the system under studied never changes. Although these assumptions may be valid to many real-time control systems, they may not be true to many practical cyber-physical systems which may exhibit multi-modal behavior property. In these systems, each mode of operation is characterized by a set of functionalities that are carried out by different update transaction sets. We call this kind of cyber-physical systems as the *dynamic cyber-physical systems (DCPS)*. A typical example is an aircraft control system given in [72]. In the system, we can distinguish landing, takeoff and normal cruise modes and each mode consists of different task sets. Another common application that is receiving increasing interests is the intelligent humanoid robots [34]. The robotic system has normal operation mode for collecting information about its working environment to decide what to do and the critical operation modes for handling urgent events.

To handle the switch among these modes, various mode change protocols have been proposed. Most of these works, however, stick to the same scheduling policy during the

entire execution of the system. They do not consider the data freshness and do not explicitly address the problem of how to maintain the temporal validity constraints during the mode change.

In this chapter we take a different approach from the past studies on maintaining the freshness of real-time data. Instead of just aiming at maintaining temporal validity as defined using validity intervals of data objects, in this chapter, our goal is to meet the temporal validity and at the same time to minimize the data staleness not only before and after the mode change in operation, but also *during* the mode changes. In general, although shorter periods for generating update jobs from each update transaction for refreshing data validity may incur heavier update cost, the staleness of the data can be better maintained. However, the tradeoff of heavier update workload could seriously affect the schedulability of the set of update transactions. In addition, normally, the sophiscated algorithms such as *DS-FP*, may incur heavier on-line scheduling overhead as they explore more scheduling information of the jobs to minimize the update workload. In addition, to address the needs of different operation modes and their resource requirements in a *DCPS*, we apply different scheduling policies in different modes based on the run-time processor workload with the purpose to minimize the data staleness if the total update workload is within the capacity of the system. We use the more neutral term, *scheduling switch*, to emphasize that a change in resource allocation policy may be in response to concerns other than the domain-specific semantics of operation modes. We aim at achieving the tradeoff between lower data staleness and better schedulability of the set of update transactions for maintaining data freshness. There are two important issues to be addressed in the problem: 1) which scheduling policy should be applied to a mode, and 2) when to conduct the switch such that the temporal validity can be maintained and the data staleness can be minimized during the transition.

To address the first problem, our strategy is to select the policies under which the set of update transactions are schedulable and we prefer the one with lower scheduling overhead and shorter update period when the system load is low to maintain higher data

freshness; On the other hand, when the system load increases, we would have to switch to more sophisticated policies that has higher schedulability, i.e., it is able to accommodate more update transactions. To maintain the temporal validity during the mode change, we propose two algorithms, named *search-based switch (SBS)* and *adjustment-based switch (ABS)* to identify proper switch points when the temporal validity constraints can be satisfied. *SBS* checks the beginning time slot of each idle period after the release of the mode change request (MCR) and verifies whether it is a proper switch point; *ABS* further relaxes the restriction on the switch point candidates and *adjusts* the schedule between the MCR and the current time slot. Compared with SBS, ABS greatly increases the number of switch point candidates and further improves the *promptness* [72] of the MCR.

The remainder of the chapter is organized as follows. Section 7.1 summarizes existing mode change protocols and reviews prior work in maintaining temporal validity in RTDBS. Section 7.2 describes our task and mode change models. Section 7.3 addresses the problem of how to decide the scheduling policies based on runtime processor workload. Section 7.4 studies the online scheduling switch problem and gets into more details with the switch between *ML* and *DS-FP*. Section 7.5 presents our performance studies and we conclude this chapter in Section 7.6.

7.1 Related Work

In the literature, mode change protocols can be classified into synchronous and asynchronous protocols with regard to the way old and new-mode tasks are combined during the mode change. [71] proposes two synchronous protocols, one with periodicity and the other without. These protocols assume that old-mode tasks may be completed if they have outstanding execution when the MCR is issued. The old-mode tasks may or may not have further jobs released depending on whether the periodicity is to be satisfied during the mode change. The offset to the MCR is obtained by summing up the worst-case execution time of all old-mode tasks.

An analysis approach for mode changes on single processor system with rate-monotonic scheduling is introduced in [77]. This protocol is based on dynamic processor utilization and the rules of the priority ceiling protocol. [87] improves and extends [77] to deadline-monotonic scheduling and with the periodicity maintained. Worst-case response time analysis is given in [87] for each type of tasks during the mode change. [65] further generalizes [87] by introducing possible offsets for all the new-mode tasks, no matter whether they are changed or unchanged tasks thus relaxes the periodicity requirement. The timing analysis in [65] assumes a pre-determined offset for each task and [72] introduces a slightly different protocol and proposes an algorithm to calculate the offsets and achieve the tradeoff between the schedulability and the promptness.

All the analysis methods in [77, 87, 65, 72] are limited to strictly periodic task activation and [37] eliminates this restriction by allowing complex task activation patterns including periodic with jitter, periodic with burst and sporadic event models. [84] further improves [37] by supporting any event stream model and it can handle both the earliest deadline first (*EDF*) and fixed priority (*FP*) scheduling of tasks.

[23] studies the mode change problem from another direction and it aims at configuring tasks within a system judiciously so that task migrations and priority changes are minimized during mode changes.

7.2 Preliminaries

7.2.1 Task and Mode Change Model

We model the operational dynamics in dynamic cyber-physical systems as a series of different modes, $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots$, and each mode $\mathcal{M}_k$ contains a fixed task set $\mathcal{T}_k = \{\tau_i\}_{i=1}^{m}$ with known C_i and $\mathcal{V}_i$ for each τ_i ($1 \leq i \leq m$). In our model, a scheduling policy Ψ_k is applied to $\mathcal{T}_k$ in mode $\mathcal{M}_k$ and following the general assumptions in the prior work, we assume that MCR is a sporadic event and it cannot occur during the mode transitions. There are four

Symbol	Definition
$\mathcal{M}_i$	The i^{th} mode in the system ($i = 0, 1, 2, ...$)
$\mathcal{T}_i$	The fixed task set in mode $\mathcal{M}_i$
Ψ_i	The scheduling policy applied on $\mathcal{T}_i$
$\mathcal{T}_k^c$	The changed task set in mode change $\mathcal{M}_k \rightarrow \mathcal{M}_{k+1}$
$\mathcal{T}_k^u$	The unchanged task set in mode change $\mathcal{M}_k \rightarrow \mathcal{M}_{k+1}$
$\mathcal{T}_k^-$	The complete task set in mode change $\mathcal{M}_k \rightarrow \mathcal{M}_{k+1}$
$\mathcal{T}_k^+$	The new task set in mode change $\mathcal{M}_{k-1} \rightarrow \mathcal{M}_k$
t_{MCR}	The issue time of the mode change request (MCR)
t_L	The latency requirement of the MCR
t_w	The switch time in the mode change
$r_{i,j}^k$	Release time of $J_{i,j}$ in mode $\mathcal{M}_k$
$d_{i,j}^k$	Absolute deadline of $J_{i,j}$ in mode $\mathcal{M}_k$

Table 7.1: Symbols and definitions

types of tasks in our model:

Complete tasks are tasks that are active in the old-mode but do not appear in the new-mode. They are allowed to run to complete under the old scheduling policy after the MCR with no new job released. They *cannot* be aborted for the purpose of maintaining its temporal validity.

New tasks are tasks that only appear in the new-mode. They are released synchronously at a proper switch point.

Unchanged tasks are tasks that are persistent through the mode change. They are executed and released after the MCR under the old scheduling policy until the new scheduling policy takes the control. The switch point should be carefully selected to maintain the temporal validity during the transition.

Changed tasks are tasks that appear in both modes but with modified parameters like C_i and $\mathcal{V}_i$ in the new-mode. The temporal validity for these tasks during the switch must also be maintained.

7.2.2 Notations and Definitions

We assume that $\mathcal{T}_k, \mathcal{T}_{k+1}$ are the two task sets before and after the mode change and they are schedulable under scheduling policies Ψ_k and Ψ_{k+1} respectively. Let $\mathcal{T}_k^c$ and $\mathcal{T}_k^u$ denote the changed and unchanged task set during the switch respectively. According to the definition in Section 7.2.1, $\mathcal{T}_k^c$ and $\mathcal{T}_k^u$ are the only tasks that appear in both modes and we have $\mathcal{T}_k^c \bigcup \mathcal{T}_k^u = \mathcal{T}_k \bigcap \mathcal{T}_{k+1}$. We further have the complete task set $\mathcal{T}_k^- = \mathcal{T}_k - \mathcal{T}_k^c - \mathcal{T}_k^u$ and the new task set $\mathcal{T}_{k+1}^+ = \mathcal{T}_{k+1} - \mathcal{T}_k^c - \mathcal{T}_k^u$.

Following tradition, we denote by t_{MCR} the time when the MCR is issued and t_L the MCR latency requirement. That is, the mode change must be conducted in $[t_{MCR}, t_{MCR}+t_L]$. Our study in this chapter focuses on the following scenario: A *DCPS* in mode $\mathcal{M}_k$ is initially controlled by Ψ_k. At time t_{MCR}, the system is notified by a change of the task set and is requested to finish a scheduling switch before time $t_{MCR} + t_L$. After t_{MCR}, the tasks in $\mathcal{T}_k^-$ are run to complete and tasks in $\mathcal{T}_k^c \bigcup \mathcal{T}_k^u$ are executed and released as normal under Ψ_k until a certain time point t_w ($t_{MCR} \leq t_w \leq t_{MCR} + t_L$) when the task set to be executed is changed to $\mathcal{T}_{k+1}$ and a new policy Ψ_{k+1} is in control. The problem is how to choose Ψ_k and Ψ_{k+1} to achieve the tradeoff between data freshness and schedulability, and how to find the valid switch point that preserves the temporal validity of the tasks in $\mathcal{T}_k^c \bigcup \mathcal{T}_k^u$ during the transition. For the reader's convenience, Table 7.1 summarizes the symbols used in this chapter but not appeared in previous chapters.

In the case that a certain task $\tau_i \in \mathcal{T}_k^c$ whose validity interval is changed from $\mathcal{V}_i$ to $\mathcal{V}_i'$ ($\mathcal{V}_i \neq \mathcal{V}_i'$) during the switch, it is difficult to identify whether the temporal validity is satisfied during the switch because it's not specified which validity interval should be used. To avoid the ambiguity, we introduce the concepts of strict and weak temporal validity as follows.

Definition 7.2.1: During the mode change from $\mathcal{M}_k$ to $\mathcal{M}_{k+1}$ and at the switch point t_w, strict temporal validity of $\tau_i \in \mathcal{T}_k^c$ is satisfied if, for the release time ($r_{i,j}^k$) of its latest job (say, the j^{th} job) that finishes before t_w under Ψ_k and the deadline ($d_{i,0}^{k+1}$) of its first job after t_w under

Ψ_{k+1}, the difference between $r_{i,j}^k$ and $d_{i,0}^{k+1}$ does not exceed the smaller validity interval. That is, $d_{i,0}^{k+1} - r_{i,j}^k \leq min\{\mathcal{V}_i, \mathcal{V}_i'\}$. Weak temporal validity is satisfied if the difference does not exceed the larger validity interval. That is, $d_{i,0}^{k+1} - r_{i,j}^k \leq max\{\mathcal{V}_i, \mathcal{V}_i'\}$ □

7.3 Utilization-Based Scheduling Selection (*UBSS*)

Prior work on mode change protocols mostly assume that the task sets before and after the mode change are controlled by the same scheduling policy. However, this assumption does not always hold in the real-world scenarios. To maintain the temporal validity in dynamic cyber-physical systems, the first problem to be addressed is how to select the proper scheduling policy for each mode so that the corresponding task set is schedulable. As we have mentioned, our strategy is to apply the periodic scheduling policies when the imposed update workload is low as long as the task set is schedulable. Because they maintain higher data freshness with lower online scheduling overhead. We only have to switch to more sophisticated policies when the schedulability bounds for periodic policies are exceeded. In this way, we achieve maintaining the temporal validity by degrading the data freshness.

In this chapter, we consider three candidate scheduling policies: *HH*, *ML* and *DS-FP*. *DS-FP* has the best schedulability among all three candidates but the worst data freshness. We choose the scheduling policy based on the imposed workload from the update transactions and the selection process is summarized in Figure 7.1. When the system is notified with a MCR, it will calculate the periods and deadlines of the new task set under *HH* and *ML*, respectively. Liu and Layland's theorem [54] is used to check the total utilization of the periodic tasks against the schedulability bound. If the task set is not schedulable under *HH*, it will be tried with *ML* instead. We will only adopt *DS-FP* when both *HH* and *ML* do not work. If the task set cannot even pass the schedulability test [33] for *DS-FP*, the system will report error because to the best of our knowledge, there is no better scheduling algorithm available than *DS-FP* for maintaining temporal validity.

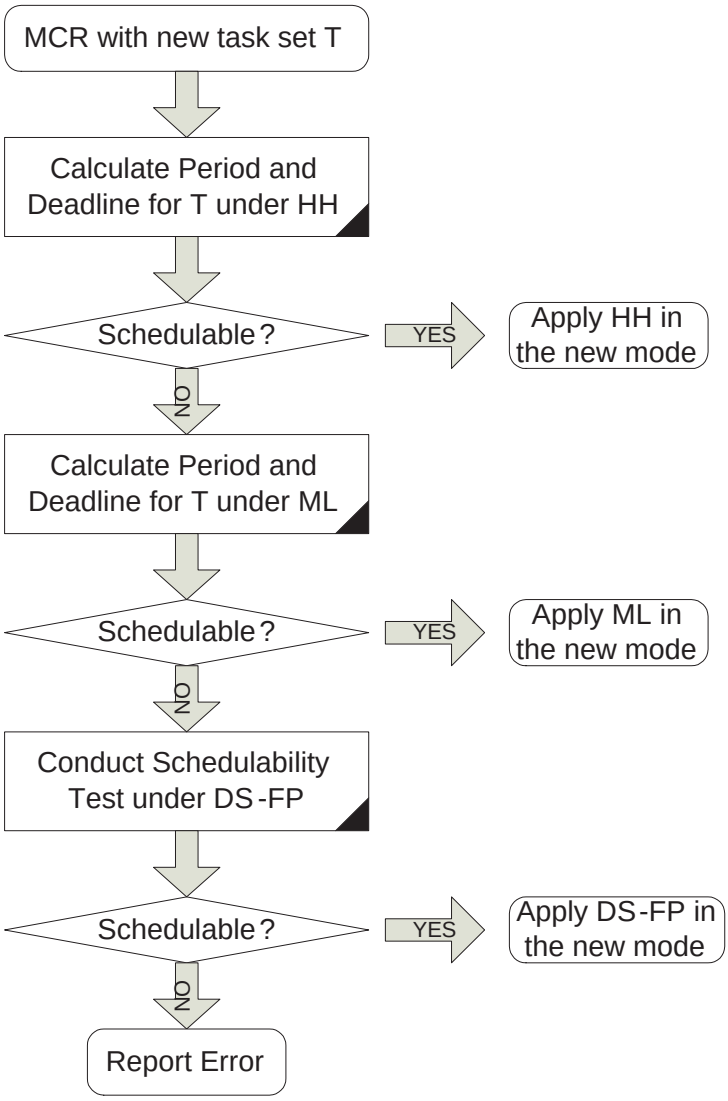

Figure 7.1: Utilization-based scheduling selection

Example 7.3.1: Consider three consecutive modes, $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$ in a *DCPS* which is shown in Figure 7.2. In $\mathcal{M}_1$, there are two transactions τ_2, τ_3 with computation times 3, 3 and validity intervals 15, 47, respectively. In $\mathcal{M}_2$, a new task τ_1 is added into the system with computation time 2 and validity interval 6. In $\mathcal{M}_3$, task τ_3's validity interval is incremented by 2 to 49 with other parameters in the task set unchanged. In $\mathcal{M}_1$, the processor utilization for the task set under *HH* is 0.525. This is below 0.828 which is the bound evaluated by Liu and Layland's theorem. According to our strategy, *HH* will be selected for scheduling in $\mathcal{M}_1$. However, in $\mathcal{M}_2$, under *ML*, the first job of τ_3, $J_{3,0}$, completes at time 24, which is greater than $\frac{\mathcal{V}_3}{2}$(that is 23.5). Thus, this new transaction set is not schedulable by either *ML* or *HH*. On the other hand, the same transaction set is schedulable by *DS-FP*, because the schedule pattern between time 26 and 50 repeats itself forever. In $\mathcal{M}_3$, with τ_3's validity interval increased to 49, the new task set is not schedulable by *HH*, but schedulable under *ML* because the deadline constraint is satisfied. □

7.4 Scheduling Switch with Validity Constraint

Even though both the old and new task sets are schedulable under the selected scheduling policies, it is not guaranteed that the temporal validity of the real-time data will be maintained. This is because the temporal validity of the tasks persistent through the switch could be violated during the scheduling switch. To satisfy all these temporal validity constraints, the switch point should be carefully selected. In this section we first investigate two different switch scenarios, the clean switch and non-clean switch. Based on the clean switch scenario, we propose two algorithms for searching the proper switch points. They are the search-based switch (SBS) and adjustment-based switch (ABS). Some theoretical results are further presented which are related to the switch between *ML* and *DS-FP*.

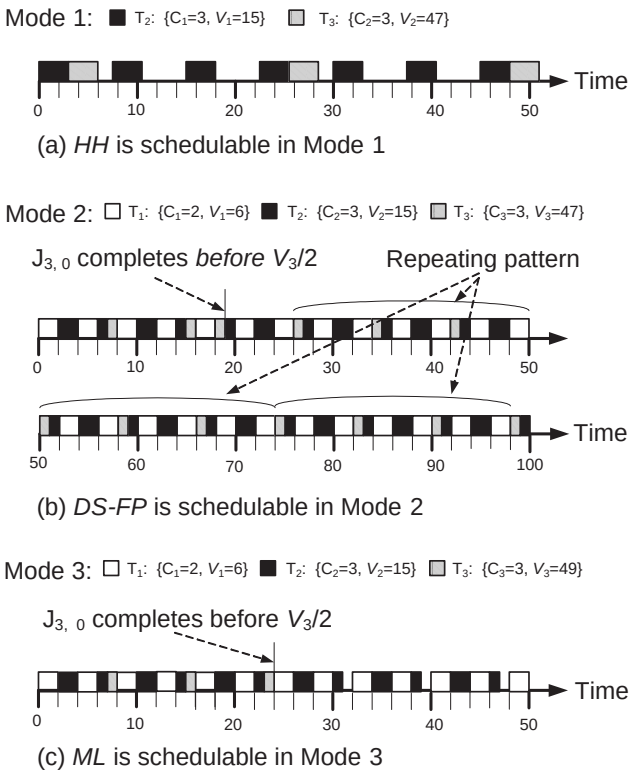

Figure 7.2: Example of utilization-based scheduling selection

7.4.1 Clean Switch vs. Non-clean Switch

During the mode change from $\mathcal{M}_k$ to $\mathcal{M}_{k+1}$, since all tasks are schedulable by their respective scheduling policies, all tasks in $\mathcal{T}_k^- \bigcup \mathcal{T}_{k+1}^+$ are schedulable and their temporal validity are also maintained. However, if a task in $\mathcal{T}_k^-$ has outstanding execution at the switch point t_w, this last job is undetermined. If we simply drop it, its temporal validity is violated; If we let it run to finish, then how it should be scheduled after t_w is unspecified.

For tasks in $\mathcal{T}_k^c \bigcup \mathcal{T}_k^u$, they are schedulable up until t_w. Ψ_{k+1} may be affected by their last jobs before t_w. Some scheduling policies are preconditioned upon the fixed starting times of the first jobs. For example all tasks start at time 0. Similar problems arise if a task in $\mathcal{T}_k^c \bigcup \mathcal{T}_k^u$ has outstanding execution at t_w. Should this job be considered the first job under Ψ_{k+1}? If so then it no longer has full execution requirement. If not then it may not meet its deadline defined in Ψ_k depending on how Ψ_{k+1} schedules it; further more, it also interferes with Ψ_{k+1} just as those tasks in $\mathcal{T}_k^-$ could do. In either case Ψ_{k+1} cannot have a clean start.

Example 7.4.1: Figure 7.3 depicts a non-clean switch of simple periodic task sets in which all tasks start at time 0 and deadline equals period. $\mathcal{T}_{k+1} = \mathcal{T}_k = \{\tau_1 = (2, 4), \tau_2 = (3, 8)\}$, $\Psi_k = \Psi_{k+1} = HH$, and $t_w = 6$. In the figure, (a) is the schedule of Ψ_k plus what could have been after t_w; (b) is the schedule of Ψ_{k+1} which treats t_w as time 0. In this run, τ_2's outstanding job is run to the finish but preempted by τ_1's first job after t_w. So it misses its deadline defined in Ψ_k. Interestingly all jobs after t_w meet deadlines in spite of this extra execution. □

The above observation is the trivial case; accordingly, we only study the switch cases where:

- The last jobs of all tasks in $\mathcal{T}_k$ are completed before t_w, i.e, there is no outstanding execution at time t_w.
- Ψ_{k+1} shall schedule $\mathcal{T}_{k+1}$ independent of the prior schedule as if t_w is its time 0.

We call such a switch scenario a clean switch; otherwise we call it a non-clean switch. In this chapter, our switch point searching algorithms will focus on the clean switch scenario. We will briefly talk about the extension to the non-clean switch scenario in Section 7.6.

7.4.2 Search-based Switch

Consider a mode change from $\mathcal{M}_k$ to $\mathcal{M}_{k+1}$. For a task in $\mathcal{T}_k^c \bigcup \mathcal{T}_k^u$ such that its last job before t_w and first job after t_w meet the deadlines in their respective Ψ, we cannot say that its real-time requirement is met because the temporal validity has not been taken into account during the switch.

Example 7.4.2: Figure 7.4 depicts a clean switch of validity constrained task sets. $\mathcal{T}_k = \{\tau_1 = (4, 16), \tau_2 = (5, 26)\}$. We switch the same task set from $\Psi_k = DS\text{-}FP$ to $\Psi_{k+1} = HH$ at $t_w = 33$. modeled in HH, $\mathcal{T}_{k+1} = \{\tau_1 = (4, 8), \tau_2 = (5, 13)\}$. In the figure, (a) is the schedule of Ψ_k plus what could have been after t_w; (b) is the schedule of Ψ_{k+1} which treats t_w as time 0. For τ_1, its last job before t_w starts at time 24 and its first job after t_w finishes at time 37. The distance is 13, which is less than $\mathcal{V}_1$. For τ_2, however, its last job before t_w starts at time 19 and its first job after t_w finishes at time 46. The distance is 27, which is bigger than $\mathcal{V}_2$. So τ_2's validity constraint is violated during the switch. □

Example 7.4.2 exposes hidden real-time requirements that could be missed. After all, all tasks meet their real-time requirements before and after the switch. What else should we consider? Introducing validity constraint has many advantages. First, it enables us to model the same real-time application in both $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$, potentially different task models. Second, it abstracts out the real-time constraints during the switch so that Ψ_k and Ψ_{k+1} could be independently applied as they are originally devised. Third, it allows the same task to fluctuate as well. For example, the execution time of a task in $\mathcal{T}_k^c$ may change to a different value after t_w.

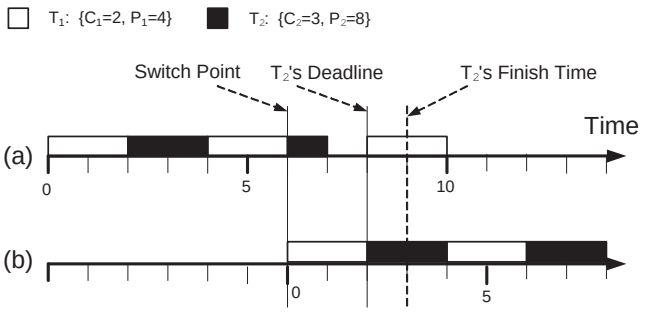

Figure 7.3: Non-clean switch from *HH* to *HH* with outstanding execution (failed)

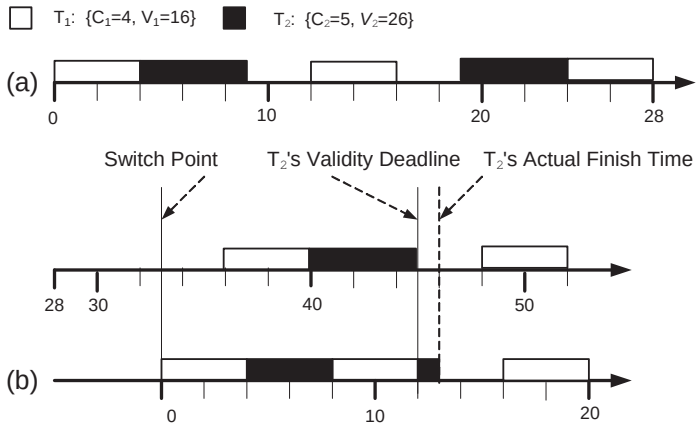

Figure 7.4: Clean switch from *DS-FP* to *HH*

We define a successful switch to be one that for all task τ_i, $\tau_i \in \mathcal{T}_k^c \bigcup \mathcal{T}_k^u$, its first job after t_w finishes within the validity interval from the start time of its last job before t_w. Our problem now becomes how to find a time point at which a successful switch is possible. This is also more realistic in practice. A dynamic cyber-physical system normally tolerates some delay (t_L) in its adjustment to the mode change. Or a successful switch point could be pre-calculated and the switch be applied before the anticipated mode change occurs.

Next we study the properties of t_w. An idle period (t_1, t_2) of a schedule begins when the last outstanding execution of any task is finished right before t_1 and there is no new job request until t_2. The process does not execute any job within (t_1, t_2). This definition

includes trivial periods in which $t_1 = t_2$. Since we are talking about clean switch, obviously t_w falls within some idle period and we have the following lemma.

Lemma 7.4.1: Any successful switch point falls in an idle period and any time point from the beginning of the idle period to this switch point is also a successful switch point.

Proof. Suppose $t_w \in (t_1, t_2)$ is a successful switch point. If we switch from any time point in (t_1, t_w), the release times of the last scheduled jobs in Ψ_k will not be changed. The new switch will shift Ψ_{k+1}'s schedule closer to time t_1 and any first job in $\mathcal{T}_k^c \bigcup \mathcal{T}_k^u$ will have an earlier finish time hence will meet the temporal validity constraint. □

Note in Figure 7.4, 33 is not a successful switch point, but time from 28 to 32 are all successful switch points. This gives us the following search-based switch algorithm for t_w.

Alg 22 Search-based Switch Algorithm

Input: $\mathcal{T}_k, \mathcal{T}_{k+1}, \Psi_k, \Psi_{k+1}$, the search start time t_0 and t_L.
Output: t_w.

```
1:  for t = t_0 to t_0 + t_L do
2:    if t == t_1 {// t_1 is the begin point of an idle period} then
3:      f = TRUE;
4:      {// Whether t is a possible candidate for t_w}
5:      for each τ_i ∈ T_k ⋂ T_{k+1} do
6:        t_s = release time of τ_i 's last job in Ψ_k;
7:        l = time to finish τ_i's first job in Ψ_{k+1};
8:        if t − t_s + l > V_i then
9:          f = FALSE;
10:       end if
11:     end for
12:     if f == TRUE then
13:       return t;
14:     end if
15:   end if
16: end for
17: return no t_w exists;
```

As all schedulable schedules have a repeating pattern in discrete time system [33], if t_L, the latency requirement of the MCR is not explicitly given, Alg. 22 runs at most the length of the shortest pattern of the schedule, $\mathcal{P}_l$, and will eventually terminate. It will report failure if there is no successful switch point. Otherwise, if there is any successful switch point, then the algorithm will always return it.

Note in the above algorithm we do not require that the parameters of a task is the same before and after the switch, which means it can apply to the tasks both in $\mathcal{T}_k^c$ and $\mathcal{T}_k^u$.

7.4.3 Adjustment-based Switch

Search-based switch (SBS) is straight forward and easy to be implemented for searching the proper switch point online. However, SBS restricts the switch point candidates only at the beginning of the idle periods in $[t_{MCR}, t_{MCR} + t_L]$. This constraint severely limits the number of possible candidates and reduces the promptness of the mode change.

We want to remove this restriction and take every time point in $[t_{MCR}, t_{MCR} + t_L]$ as the candidate for scheduling switch. However, this extension will put us in the non-clean switch scenario because there could be outstanding execution for certain jobs at that time. In this section, we propose the adjustment-based switch algorithm (ABS) to address this problem. ABS converts the non-clean switch scenario to clean switch scenario through schedule adjustment. By schedule adjustment, we mean changing release times and deadlines of jobs. The basic idea of ABS is, at a certain time t ($t_{MCR} \leq t \leq t_{MCR}+t_L$), we push all the outstanding execution back to t. That is, all the unfinished jobs in $\mathcal{T}_k$ must be finished by t in the adjusted schedule. Then the schedule in $[t_{MCR}, t]$ will be adjusted backwards from time t so that the adjusted schedule is valid for guaranteeing the validity constraints for all update transactions. Notice that if transaction τ_h is the highest priority transaction in $\mathcal{T}_k$ whose schedule needs to be adjusted, then the schedule of all lower-priority transactions $\tau_i (h < i \leq m)$ in $\mathcal{T}_k$ also needs to be adjusted due to the impact of release time and deadline adjustment of τ_h. After the schedule adjustment, ABS will release all the tasks in $\mathcal{T}_{k+1}$ at

time t and checks whether for each task $\tau_i \in \mathcal{T}_k^c \bigcup \mathcal{T}_k^u$, it can maintain the temporal validity during the scheduling switch.

Alg 23 Adjustment-based Switch Algorithm

Input: $\mathcal{T}_k, \mathcal{T}_{k+1}, \Psi_k, \Psi_{k+1}, t_0$ and t_L.
Output: t_w.

```
1:  for t = t_0 to t_0 + t_L do
2:    {// ∑_i Γ_i(t) is accumulated outstanding execution at t}
3:    if I(t_0, t) < ∑_i Γ_i(t) then
4:      continue;
5:    else {// Adjust the schedule of T_k in [t_0, t).}
6:      f = ScheduleAdjustment (T_k, t_0, t);
7:      if f == FAIL then
8:        continue;
9:      else
10:       for each τ_i ∈ T_k ⋂ T_{k+1} do
11:         t_s = adjusted request time of τ_i 's last job in Ψ_k;
12:         l = time to finish τ_i's first job in Ψ_{k+1};
13:         if t − t_s + l > V_i {// The temporal validity is violated.} then
14:           f = FAIL;
15:         end if
16:       end for
17:       if f == SUCCESS then
18:         return t;
19:       end if
20:     end if
21:   end if
22: end for
23: return no t_w exists;
```

The framework of the adjustment-based switch is presented in Alg. 23. We denote by $I(a, b)$ the total number of idle slots between $[a, b]$, and $\Gamma_i(t)$ the outstanding execution of τ_i at time t. In Alg. 23, line 3-5 checks that at each candidate time t, whether there are enough idle slots in $[t_0, t]$ to accommodate all the outstanding execution. Line 8 is the core of the algorithm. The function *ScheduleAdjustment* (T, t_0, t) tries to push back all the outstanding execution of transaction set T at time t and adjust the schedule in $[t_0, t]$ to satisfy the temporal validity constraints. Alg. 24 presents the details of this adjustment

Alg 24 ScheduleAdjustment (T, t_0, t)

Input: Transaction set T and adjustment period $[t_0, t]$.

Output: Adjusted schedule in $[t_0, t]$ and $\forall \tau_i$, the adjusted release time of its last job before t.

$h = \min_i\{i | \tau_i \in T$ and τ_i has outstanding execution at $t.\}$
$\forall i < h, k_i = k_i - 1$; {// No adjustment for $i < h$}
for $i = h$ **to** m **do**
  $d'_{i,k_i} = t$; {// d'_{i,k_i} is adjusted from d_{i,k_i}.}
  $j = k_i$; {// J_{i,k_i} is the last job of τ_i in $[t_0, t]$}
  $t_s = t$; {// Schedule in $[t_s, t]$ will be adjusted.}
  while $(j > 0)$ **do**
    if $(d'_{i,j} - r_{i,j} < \Theta_i(r_{i,j}, d'_{i,j}) + C_i)$ {// $J_{i,j}$'s response time $> d'_{i,j} - r_{i,j}$} **then**
      $r'_{i,j} = d'_{i,j} - \Theta_i(r'_{i,j}, d'_{i,j}) - C_i$;
      if $(((j < k_i) \wedge (d'_{i,j+1} - r'_{i,j} > \mathcal{V}_i)) \vee (r'_{i,j} < t_0))$ {// adjustment fails} **then**
        return FAIL;
      end if
      if $((r'_{i,j} < d_{i,j-1}))$ **then**
        $d'_{i,j-1} = r'_{i,j}$;
      else
        $d'_{i,j-1} = d_{i,j-1}$;
      end if
      $j = j - 1$;
    else {// No adjustment for this job}
      if $(d'_{i,j} - \Theta(t_0, d'_{i,j}) - C_i < t_0)$ **then**
        return FAIL; {// cannot adjust the schedule before t_0}
      else
        if $(t_s \geq d'_{i,j})$ **then**
          $t_s = d'_{i,j}$;
          break;
        else
          $d'_{i,j-1} = d_{i,j-1}$;
          $j = j - 1$;
        end if
        if $((j == 0) \wedge (d'_{i,j} \neq d_{i,j}))$ **then**
          return FAIL;
        end if
      end if
    end if
  end while
end for
return adjusted S in $[t_0, t]$ and $\forall i, r'_{i,k_i}$;

process. If the adjustment is successful, line 13-18 further verifies whether the temporal validity is also maintained during the switch. Alg. 23 sequentially checks each time slot in $[t_0, t_0 + t_L]$ and it returns the earliest proper switch point or reports failure when time $t_0 + t_L$ is reached.

Alg. 24 summarizes the details of the schedule adjustment. In the algorithm, line 2 first identifies τ_h, the transaction with the highest priority in $\mathcal{T}_k$ who has outstanding execution at time t. For each transaction τ_i ($h \leq i \leq m$), line 7 assigns t as the deadline of its last job in $[t_0, t]$ if it has outstanding execution. Alg. 24 adjusts the release time and deadline for each job of these transactions backward sequentially until the condition in line 27 is satisfied where no further adjustment is needed. With the adjusted deadline, line 11 checks whether the corresponding job can be scheduled without exceeding the original release time. Line 13 adjusts the job's release time if necessary. Line 15 verifies two important conditions: 1) whether the job's release time is pushed back before t_0 and, 2) whether the temporal validity is still maintained after the adjustment. Failure will be reported if either of the conditions is not met. Similar checking will also be conducted in line 24 even when no adjustment is conducted. Line 17-21 adjusts the deadline of the previous job for further processing. If a successful adjustment cannot be achieved even when all the jobs are tested, line 35 will report failure. Otherwise, the successfully adjusted schedule will be returned.

Example 7.4.3 shows a scenario where the adjustment-based switch outperforms the search-based switch.

Example 7.4.3: Following the same task set as in Example 7.4.2, Figure 7.5 depicts a scenario where search-based switch does not work while adjustment-based switch is successful. The MCR is issued at time 33 and its latency requirement is 12 which means the scheduling switch must be finished before time 45. The only switch candidate under search-based switch is time 33 but it does not satisfy the validity constraint for τ_2 during the transition. However, if adjustment-based switch is applied at time 42, the outstanding execution of τ_2 is 3 and they can be adjusted to $[33, 36]$ for execution. Under adjustment-

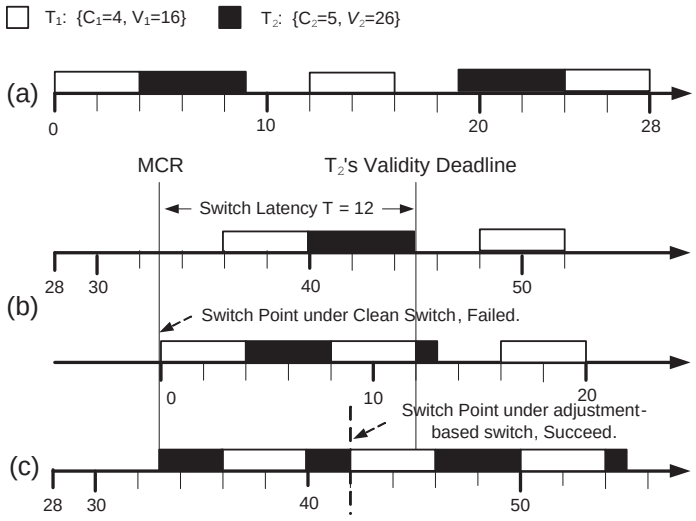

Figure 7.5: An example of successful adjustment-based switch

based switch at time 42, in the adjusted schedule, τ_2's last job before time 42 is 33 and its first job after time 42 is 55. This distance is 22 which is smaller than $\mathcal{V}_2$. So the validity constraint during the switch is satisfied. □

7.4.4 Properties of Switch between *ML* and *DS-FP*

In this section, we investigate further and look at the switch specifically between *ML* and *DS-FP*. *HH* can be taken as a special case of *ML* with $P_i = D_i = \frac{\mathcal{V}_i}{2}$. Theorem 7.4.1 and Theorem 7.4.2 presents two interesting properties of the switch between them.

Theorem 7.4.1: For $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$, if $\Psi_k = \Psi_{k+1} = ML$, then any idle point is a successful switch point.

Proof. We prove that the validity constraint of any task τ in $\mathcal{T}_k^c \bigcup \mathcal{T}_k^u$ is met during the switch. Let $\tau = (C, P)$ and t_w is any idle time considered for switch. Let t be the release time of τ's last job before t_w. Since t_w is an idle time, we have $t_w - t \leq P$. From t_w, the first job of τ will be finished within D. So the distance from the release time of the last job under

Ψ_k to the finish time of the first job under Ψ_{k+1} is no more than $(t_w - t) + D \leq P + D = \mathcal{V}$. □

Note Theorem 7.4.1 holds when $\mathcal{T}_k \neq \mathcal{T}_{k+1}$. It also holds if the execution time of tasks in $\mathcal{T}_k^c \bigcup \mathcal{T}_k^u$ changes during the switch as long as *ML* schedules both $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$. If validity interval also changes, we then have to go back to Alg. 22 for searching successful switch points to maintain strict temporal validity.

Theorem 7.4.2: For $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$, if $\mathcal{T}_{k+1} \subseteq \mathcal{T}_k$, $\Psi_k = ML$ and $\Psi_{k+1} = DS\text{-}FP$, any idle point is a successful switch point.

Proof. We prove that the validity constraint of any task τ in $\mathcal{T}_{k+1}$ is met during the switch. Let t_w be any idle time considered for the switch. Let t be the release time of τ's last job before t_w. Since t_w is an idle time, we have $t_w - t \leq P$. Since $\mathcal{T}_k$ is schedulable by *ML*, so is its subset $\mathcal{T}_{k+1}$. So the worst-case execution time of τ under *ML* is no bigger than D. Furthermore, according to Theorem 6.2.2, the worst-case execution time of τ under *DS-FP* is no bigger than D. So the first job of τ after t_w finishes within D. So the distance from the release time of the last job under Ψ_k to the finish time of the first job under Ψ_{k+1} is no more than $(t_w - t) + D \leq P + D \leq \mathcal{V}$. □

Note that theorem 7.4.2 may not hold when $\mathcal{T}_{k+1} \nsubseteq \mathcal{T}_k$. Even if $\mathcal{T}_{k+1} \subseteq \mathcal{T}_k$, we cannot extend the result to cases where the execution time or validity interval changes. For $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$, if $\Psi_k = DS\text{-}FP$ and $\Psi_{k+1} = ML$, an idle time point may or may not be a clean switch point. This is true even if $\mathcal{T}_k = \mathcal{T}_{k+1}$, which is already demonstrated in Example 7.4.2.

7.5 Performance Evaluation

This section presents simulation results of the online scheduling switch in dynamic cyber-physical systems. We present two sets of experiments for the performance evaluation. The first set of experiments focuses on the comparison between single scheduling policy and

online utilization-based scheduling switch (*UBSS*). The second set compares the ability and efficiency of the two algorithms, search-based switch (SBS) and adjustment-based switch (ABS), for searching proper switch points. Our goal is to study the efficiency of the invented algorithms and verify that *UBSS* can maintain higher data freshness and significantly reduce the online scheduling overhead while satisfying the temporal validity constraints over the entire execution of the system.

7.5.1 Simulation Model and Parameters

Two categories of parameters are defined: system and update transaction parameters. For system configurations, we consider a single CPU, main memory based RTDBS. There are up to 10 modes in the system. The number of real-time data objects in the system varies from 1 to 20 and the validity length is uniformly distributed from 50 to 150 time units. It is assumed that each transaction updates one real-time data object, and its CPU time is uniformly distributed from 1 to 5 time units. Following the definition of [97], we define the *density factor* of a set of transactions $\mathcal{T}$, denoted by γ, as $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i}$. The primary performance metrics used in our experimental studies are the CPU utilization, the scheduling success ratio, the data staleness, the scheduling overhead and the switch latency.

7.5.2 Performance Improvement with *UBSS*

In this set of experiments, we simulate the fluctuation of the system as a sequence of 10 consecutive modes and each mode has a fixed duration of 20000 time units. The density factor for each mode is shown at the top of Figure 7.6. 200 task sets are randomly generated in each mode and the scheduling success ratios for *ML*, *DS-FP* and *UBSS* are evaluated respectively. The comparison is illustrated at the bottom of Figure 7.6. In the figure, our first important observation is, *DS-FP* and *UBSS* always have the same success ratio in different modes with varying density factors. The reason is at the beginning of each mode, *UBSS* will conduct a schedulability test and select the most proper policy for scheduling. If a task set

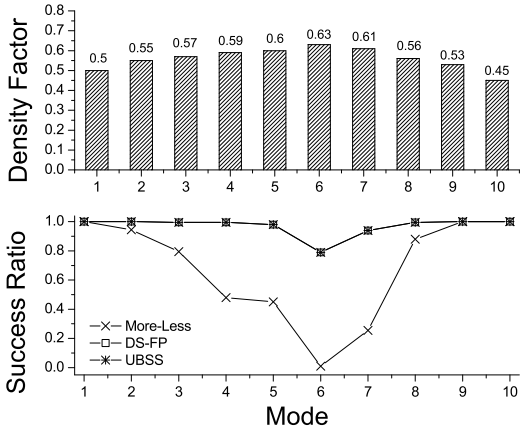

Figure 7.6: Success ratio vs. CPU util

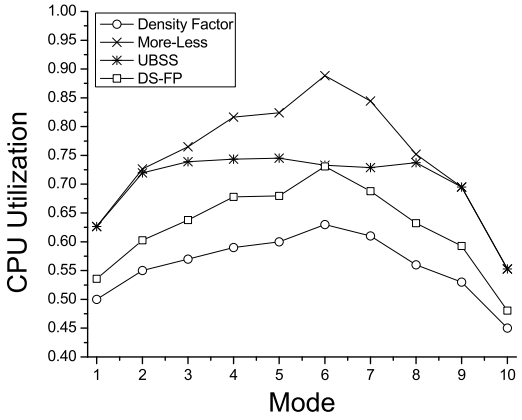

Figure 7.7: Comparison of CPU utilization

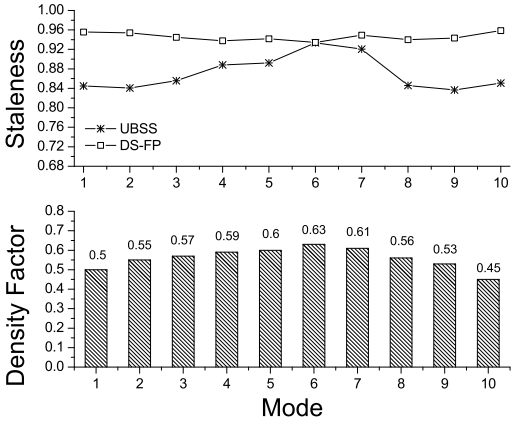

Figure 7.8: Staleness with high workload

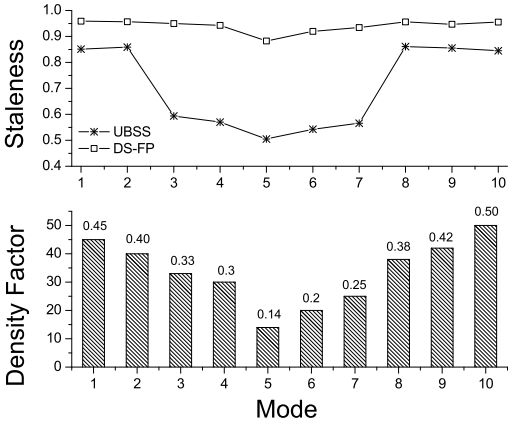

Figure 7.9: Staleness with low workload

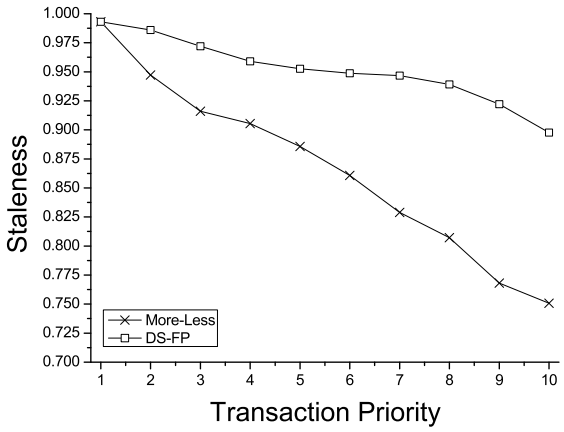

Figure 7.10: Staleness vs. Priority

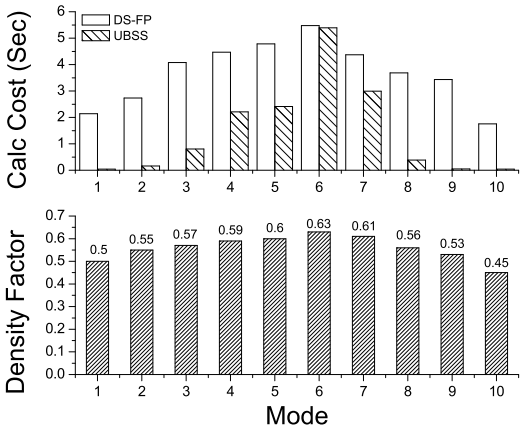

Figure 7.11: Overhead vs. CPU utilization

is only schedulable under *DS-FP*, *UBSS* will choose *DS-FP* for the purpose of maximized schedulability. In Figure 7.6, we also observe that the success ratios of all three approaches drop along with the increase of the density factor while *DS-FP* and *UBSS* outperform *ML* persistently through the whole system. The success ratio of *ML* drops to 0.01 when the density factor climbs to 0.63 while *DS-FP* and *UBSS* can still maintain a 0.79 success ratio. All three approaches have the same success ratio when the density factor is below 0.55 because all task sets are schedulable under *ML* at that time.

Figure 7.7 shows the comparison of the CPU utilization among three approaches. As mentioned before, *DS-FP* can greatly reduce the CPU utilization compared with *ML* while still maintaining the temporal validity. This is verified in Figure 7.7 where the CPU utilization of *DS-FP* is consistently lower than *ML* and the difference reaches 15.7% when the density factor is 0.63 in mode 6. As observed in Figure 7.7, the CPU utilization of *UBSS* is between the *ML* and *DS-FP*. When the density factor is low, the CPU utilization of *UBSS* is close to *ML* because most of the task sets are schedulable under *ML* at that time and *UBSS* prefers choosing periodic scheduling policy for maintaining higher data freshness and lower online scheduling overhead. On the other hand, when the system workload is high and most

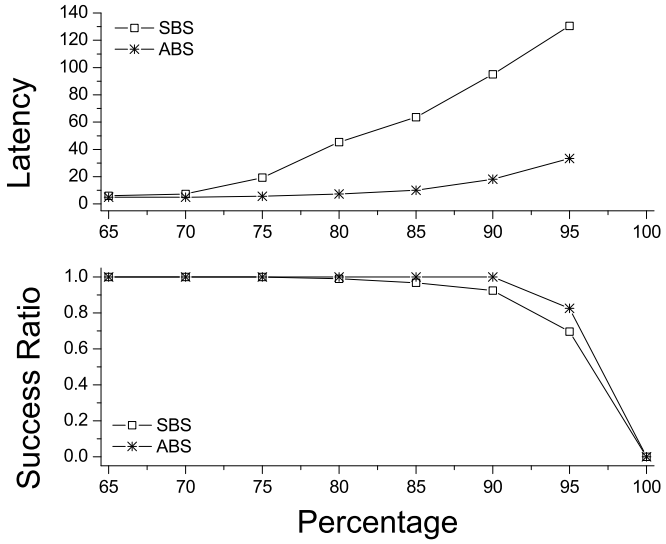

Figure 7.12: Comparison of the switch latency

task sets are only schedulable under *DS-FP*, *UBSS* will let *DS-FP* take control and achieve the maximized schedulability. Its CPU utilization will be close to *DS-FP* at that time.

Figure 7.8 and Figure 7.9 demonstrate *UBSS*'s improvement over *DS-FP* in maintaining higher data freshness with different system workloads. Suppose mode $\mathcal{M}_k$ contains the task set $\mathcal{T}_k = \{\tau_i\}_{i=1}^{m}$ and τ_i $(1 \leq i \leq m)$ has released N_i update jobs by the end of $\mathcal{M}_k$. We measure $\overline{S}_k$, the average data staleness of $\mathcal{T}_k$ by summing up each update job $(J_{i,j})$'s staleness at their finishing time $(f_{i,j})$ and divided by the total number of updates, i.e., $\overline{S}_k = \frac{\sum_{i=1}^{m}\sum_{j=0}^{N_i} S_t(f_{i,j})}{\sum_{i}^{m} N_i}$. Both Figure 7.8 and Figure 7.9 show that the average data staleness under *UBSS* is always lower than *DS-FP* which means the real-time data values under *UBSS* is persistently fresher. Figure 7.9 further shows that the improvement increases when the system workload decreases and it reaches around 40% when the density factor is 0.14 in mode 5. The main reason for this huge improvement is, when the system workload is low, the transaction set has a high possibility to be schedulable under *ML* or even *HH* where the deadline and period for each transaction are both assigned to be half of the validity length.

This assignment greatly shortens the data staleness thus improve the data freshness. On the other hand, *DS-FP* always defers the release time of the update jobs as late as possible. This drives its data staleness close to the validity length and both Figure 7.8 and Figure 7.9 show that the data staleness remains around 95% and is quite stable in the presence of the fluctuation of the system workload. Figure 7.10 helps us gain an insight into the comparison of data staleness between *ML* and *DS-FP* in mode 5. We have several observations in this figure. First, for each update transaction, the real-time data value under *ML* is consistently fresher than that of *DS-FP* and the transaction with lower priority has larger improvement; Second, transactions with lower priorities always have lower staleness no matter which scheduling policy is employed.

Figure 7.11 compares the online scheduling overhead between *DS-FP* and *UBSS* in each mode. From the figure, we observe that with the same scheduling success ratio, *UBSS* can greatly reduce the online scheduling overhead especially when the density factor is low. For example, in mode 10, when the density factor is 0.45, the scheduling overhead of *DS-FP* is 1.75369 which is 41.2 times higher than that of *UBSS* (0.04257). This is because in those scenarios, *ML* will be in control most of the time and its online scheduling overhead is much lower than that of *DS-FP*.

7.5.3 Search-based Switch vs. Adjustment-based Switch

In this subsection, we compare the efficiency of the two algorithms for searching switch points from *DS-FP* to *ML* switch scenario. The task sets before and after the switch, $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$, are simulated as follows. With fixed density factor ($\gamma = 0.6$) and transaction number ($m = 20$), we randomly generate $\mathcal{T}_k = \{\tau_i\}_{i=1}^{m}$ and make sure that $\mathcal{T}_k$ is only schedulable under *DS-FP*. $\mathcal{T}_{k+1}$ is defined as a subset of $\mathcal{T}_k$ and is specified by a given percentage p. $\mathcal{T}_{k+1}$ contains the first $\lceil m \times p\% \rceil$ transactions with higher priorities in $\mathcal{T}_k$, i.e., $\mathcal{T}_{k+1} = \{\tau_i\}_{i=1}^{\lceil m \times p\% \rceil}$. We set the switch latency as 2000 time units and compare the success ratio and switch latency between the two algorithms. We conduct 200 experiments for each point to

get the average value.

In Figure 7.12, we observe that the switch success ratio under SBS and ABS are both decreasing when $\mathcal{T}_{k+1}$ increases its size. This is because the more transactions $\mathcal{T}_{k+1}$ has, the more temporal validity constraints to be applied on the switch point candidates. On the other hand, by proactively *creating* the switch point through schedule adjustment while not only searching passively, ABS always outperforms SBS in terms of the switch success ratio and the difference reaches 13% when the percentage p climbs to 95%. As $\mathcal{T}_k$ is not schedulable under *ML*, when $p = 100\%$, $\mathcal{T}_{k+1}$ is equal to $\mathcal{T}_k$ and the success ratio for both of them drops to 0. Figure 7.12 also shows the comparison of the switch latency between SBS and ABS. We can see that ABS always has lower switch latency and the improvement keeps increasing when $\mathcal{T}_{k+1}$ increases. This is because SBS restricts the switch candidates only at the beginning time slot of idle periods while ABS breaks this constraint. Through judicious schedule adjustment, ABS greatly increases the number of candidates for scheduling switch and potentially improve the switch latency.

7.6 Summary

In this chapter we studied the problem how to maintain the temporal validity of real-time data in the presence of mode changes in dynamic cyber-physical systems. We proposed to use different scheduling policies in different modes and introduced two algorithms to search for proper switch points. We studied the properties of the switch point and provided some results on switching between two scheduling policies, *ML* and *DS-FP*. Extensive experiments are conducted to evaluate the algorithm performance and show that switch between different scheduling policies according to runtime processor workload can significantly outperform a single fixed scheduling policy while only introduce limited switch overhead.

Chapter 8

Maintaining Data Freshness and Control Quality in Cyber-Physical Sensing and Control Systems

In Chapter 6 and 7, we presented a family of scheduling algorithms for guaranteeing the validity of real-time data in both static and dynamic cyber-physical systems. A cyber-physical system, however, is typically a real-time sensing and control system where in addition to maintaining a set of update transactions for updating the measurements of the physical entities in the operating environment, there is another type of transactions called control transactions. Control transactions are defined according to specific sensing and control requirements. They access to the real-time data objects maintained in the real-time database for detecting critical events occurring in the operating environment, and taking corresponding actions. For example, in the cyberphysical avatar, control transactions may be defined to monitor the measurements of the force torque sensors and use the force-feedback signals for controlling the series elastic actuators (SEA) employed at all joints of the robot hands. Once a certain pre-defined condition is reached, a control decision will be generated by the control transaction and be submitted to designated actuators. To perform monitoring func-

tions effectively, the control transactions should be invoked periodically. Each invocation is called a control job which has a hard or firm deadline on its completion time. Missing the deadline could severely degrade the system performance, i.e., it cannot generate a timely response to react to a critical event. Thus, in the design of these systems, it is important to meet the deadlines of all the control jobs using a real-time scheduling algorithm. Furthermore, if the control transactions access to stale data [27, 68], the effectiveness of the monitoring functions provided by them will also be seriously degraded. Thus, another important issue to ensure the effectiveness in event monitoring is to maintain the freshness of the real-time data objects in a real-time database as well.

Although the algorithms presented in previous chapters, e.g., *ML*, *DS-FP* and *DS-LALF*, are effective in maintaining the validity of real-time data, they have ignored the impact on the performance of the control transactions. Obviously, the co-scheduling of these two types of transactions is conflicting with each other as both of them need to meet the deadlines and at the same time compete with each other for the same set of resources for processing. Intuitively, assigning higher priorities to the update transactions can maximize the quality of the real-time data objects, while the tradeoff is that the deadline constraints of the control transactions may not be satisfied. On the other hand, scheduling the control transactions first can maximize their chances to meet the deadlines, but the quality of the real-time data objects can be seriously degraded. This could seriously hurt the effectiveness and correctness of the monitoring functions to be performed by the control transactions.

The previous studies on solving the co-scheduling problem of update and control transactions mainly focus on soft real-time systems. In [66], a two-level co-scheduling algorithm called query update time sharing (QUTS) is proposed. The co-scheduling of updates and queries is also studied in [86]. However, unlike [66], it is for systems where the arrival of updates and queries are pre-defined. Based on the update first and application first methods, it proposes an optimal schedule for a given set of updates and queries to maximize both quality of data and quality of services. In [46], the evaluation of application

queries over dynamic update streams is studied and a load-adaptive scheme is proposed based on the rate monotonic scheduling. In [14, 44, 42], a set of control-based methods are proposed to deal with unpredictable system workload and changing data requirements of real-time transactions. They apply a feedback-based miss ratio control scheme to maintain the performance within a desired value.

Unlike these previous works on co-scheduling for soft real-time systems, in this chapter, we aim to provide a *guarantee* in performance in meeting the deadline constraints of the control transactions and at the same time to maximize the quality of data objects for execution of the control transactions. We extend *DS-LALF* proposed in Section 6.5 to be a dynamic co-scheduling algorithm, called *Co-LALF*. The performance goal of *Co-LALF* is to construct a schedule that can meet the deadlines of all the periodic control transactions and at the same time maximize the quality of data (QoD) of the data objects for execution of the control transactions. *Co-LALF* has three novel features: 1) similar to *DS-FP* and *DS-LALF*, it adopts the aperiodic task model for generating update jobs and defers their release times as much as the corresponding real-time data objects are still valid defined according to their minimum validity intervals. This mechanism helps reduce the CPU utilization imposed from the update transactions and thus improve the schedulability of the control transactions without sacrificing the QoD of the real-time data objects; 2) *Co-LALF* applies a dynamic priority assignment technique and always schedules the job which is the most urgent in maintaining the system schedulability. In this way, *Co-LALF* can provide a finer granularity in co-scheduling update and control transactions; and 3) *Co-LALF* extends the validity intervals of the update transactions within their validity ranges when the control jobs cannot be scheduled. This adjustment offers flexibility in co-scheduling of update and control transactions, and achieves a good balance between the system schedulability and the QoD of data objects. Our experimental results in Section 8.3 validate the effectiveness of *Co-LALF* in improving the system schedulability and providing better QoD for the control transactions.

Symbol	Definition
$\mathcal{X}_i$	Real-time data object i
$\mathcal{T}^u/\mathcal{T}^c$	The set of update / control transactions
τ_i^u	Update transaction i for updating $\mathcal{X}_i$
τ_i^c	Control transaction i
$\mathcal{V}_i^{min}/\mathcal{V}_i^{max}$	Minimum / Maximum validity interval of $\mathcal{X}_i$
$\mathcal{V}_i$	Selected validity interval of $\mathcal{X}_i$
$J_{i,j}^u/J_{i,j}^c$	The $j+1^{th}$ job of task τ_i^u/τ_i^c $(j = 0, 1, 2, \ldots)$
C_i^u/C_i^c	Worst-case execution time (WCET) of τ_i^u/τ_i^c
D_i^u/D_i^c	Relative deadline of τ_i^u/τ_i^c
d_i^u/d_i^c	Deadline of τ_i^u/τ_i^c
P_i^u/P_i^c	Period of τ_i^u/τ_i^c
$rc_{i,j}^u/rc_{i,j}^c$	The remaining execution time of $J_{i,j}^u$ / $J_{i,j}^c$
$r_{i,j}^u/r_{i,j}^c$	The release time of $J_{i,j}^u/J_{i,j}^c$
$S_i(t)$	The staleness of data object $\mathcal{X}_i$ at time t
$Q_i^u(t)$	The quality of data (QoD) of $\mathcal{X}_i$ at time t

Table 8.1: Symbols and definitions

The remainder of the chapter is organized as follows. In Section 8.1, we present the system model, and define the quality of data (QoD) for real-time data objects and the quality of control (QoC) provided by the control transactions. Section 8.2 presents the details of the *Co-LALF* algorithm for tackling the update and control transactions' co-scheduling problem. Section 8.3 reports the performance of *Co-LALF*, and we conclude this chapter in Section 8.4.

8.1 System Model, Quality of Data & Control

In this section, we first introduce our system model of Cyber-Physical sensing and control systems. We then define the quality of data (QoD) for a real-time data object and the quality of control (QoC) for a control transaction. Table 8.1 summarizes the set of symbols that are frequently used in the chapter.

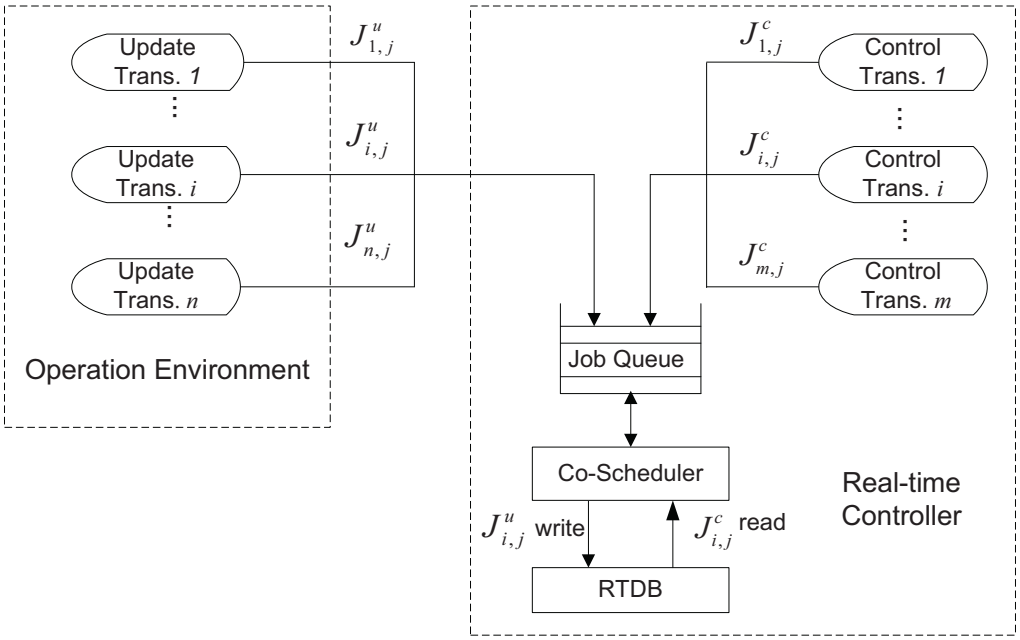

Figure 8.1: Conceptual system model

8.1.1 System Model and Assumptions

Figure 8.1 depicts the conceptual system model of a real-time sensing and control system to be studied in this chapter. The system consists of a fixed set of periodic control transactions $\mathcal{T}^c$ and a set of update transactions $\mathcal{T}^u$. The control transactions are defined at the real-time controller according to the application-dependent control strategies while the update transactions are defined one for each real-time data object maintained in the real-time database (RTDB) for installing the corresponding sensor measurements into the database. The database is assumed to be maintained in the main memory.

Each update transaction τ_i^u is responsible for maintaining the validity of a real-time data object X_i. For each update value of X_i, τ_i^u releases an update job $J_{i,j}^u (j = 0, 1, 2, \ldots)$ to install it into the real-time database to refresh the validity of X_i. The release time of an

update job is the time when it is generated by capturing the current status of a dynamic entity in the operational environment. We use C_i^u to denote the worst-case execution time (WCET) for the jobs generated from τ_i^u. Although the transmission time for an update job from τ_i^u to the real-time controller may not be a constant due to variations of transmission delays in the network, it is assumed to be bounded by δ_i. Note that if the value of δ_i is larger, the cost for maintaining the data validity will be higher as the period for generating the update jobs will be shorter [95]. Due to the transmission delay of an update job to the controller and scheduling delay, the completion time of an update job could be much greater than the release time of the job. To maintain the data validity, it is important to ensure the completion time of an update job to be earlier than its deadline.

Each control transaction, τ_i^c, is characterized by a fixed period P_i^c, a worst-case execution time (WCET) C_i^c, a relative deadline, D_i^c, and an update transaction set Ω_i. τ_i^c generates a control job, $J_{i,j}^c (j = 0, 1, 2, \ldots)$, at the beginning of every period P_i^c with the job deadline set to be its release time plus the relative deadline, D_i^c. Each control job consists of a set of pre-defined read operations. Based on the values of the accessed data objects, it generates a control decision according to the defined monitoring function to respond to the event occurring in the operational environment. Ω_i defines the set of update transactions that refreshes the set of data objects to be accessed by τ_i^c.

At the real-time controller, the received update jobs together with the released periodic control jobs are sorted according to their priorities in the job queue. The co-scheduler selects the highest priority job from the job queue for processing and the scheduling is preemptive. If a higher priority job is released and inserted into the job queue, the current job that is processing will be preempted and returned into the job queue.

8.1.2 A Range of Validity Intervals

Instead of using a single data validity interval for a real-time data object as in Chapter 6 and 7, in this chapter, we extend the concept and adopt a range of validity intervals $[\mathcal{V}_i^{min}, \mathcal{V}_i^{max}]$

to define the validity constraints for data object $\mathcal{X}_i$. $\mathcal{V}_i^{min}$ is the minimum validity interval of $\mathcal{X}_i$ and the corresponding update transaction τ_i^u. The definition of $\mathcal{V}_i^{min}$ is similar to the definition of validity interval in Definition 6.1.1. However, it is assumed that after $\mathcal{V}_i^{min}$, although a data object is considered to be stale, it still has some "value" to the control transactions. An update job will be considered to be totally useless after the maximum validity interval $\mathcal{V}_i^{max}$. The validity interval $\mathcal{V}_i$ for maintaining the validity of $\mathcal{X}_i$ is determined within the range, *i.e.*, $\mathcal{V}_i^{min} \leq \mathcal{V}_i \leq \mathcal{V}_i^{max}$.

The benefit of defining a range of validity intervals for a real-time data object is that it can provide flexibility in update scheduling for trading off between real-time data quality and system schedulability. In cases, especially in the co-scheduling scenarios, when we cannot satisfy the deadline constraints of all the control transactions while ensuring that all the data objects are valid using the minimum validity intervals, we may adjust the schedule using larger validity intervals for the data objects provided that they are not larger than the corresponding maximum validity intervals. Notice that in the scheduling algorithms introduced in Chapter 6, *e.g.*, *DS-FP* and *DS-LALF*, guaranteeing the validity of the real-time data objects is the most important performance objective. Thus, all the control transactions are assigned lower priorities compared with the update transactions and the validity intervals of the data objects are always set to be their minimum validity intervals to maximize the quality of the data. This assumption will be removed when we are studying the update and control co-scheduling problem in this chapter.

8.1.3 Quality of Data (*QoD*) and Quality of Control (*QoC*)

Following the definition of data validity in Section 8.1.2, we define the *staleness* of a real-time data object.

Definition 8.1.1: Assume that the current value of data object $\mathcal{X}_i$ is sampled at time t_s and its validity interval is $\mathcal{V}_i$ which is set to be $\mathcal{V}_i^{min}$. The staleness of $\mathcal{X}_i$ at time t is defined as:

$$S_i(t) = \begin{cases} 0, & t \in [t_s, t_s + \mathcal{V}_i^{min}] \\ t - (t_s + \mathcal{V}_i^{min}), & t > t_s + \mathcal{V}_i^{min} \end{cases}$$

Note that under this definition, the minimum value of staleness of a data object is zero while its maximum value is unbounded and increases with time linearly after $\mathcal{V}_i^{min}$.

Observing stale data objects can make a control job generate inaccurate or even harmful result. Therefore, in addition to meeting their deadlines, another important performance objective is to maximize the *quality of data (*QoD*)* of the data objects observed by a control job. Different sensing and control applications may have different functions for quantifying the quality of a data object as a function of time after $\mathcal{V}_i^{min}$. In this chapter, to simplify the discussion, we define the *quality of data* (*QoD*) of data object $\mathcal{X}_i$ at time t as follows:

$$\mathcal{Q}_i^u(t) = 1 - \frac{\min\{S_i(t), \mathcal{V}_i^{max} - \mathcal{V}_i^{min}\}}{\mathcal{V}_i^{max} - \mathcal{V}_i^{min}} \tag{8.1}$$

where $S_i(t)$ is the staleness of $\mathcal{X}_i$ at time t defined according to Definition 8.1.1. According to Eq. 8.1, the maximum value of the *QoD* of $\mathcal{X}_i$ is bounded by 1 because the staleness is no smaller than 0. If the *QoD* of $\mathcal{X}_i$ is between 0 and 1, then $\mathcal{X}_i$ still has some "value" to the control transactions. Otherwise, it is totally useless or even harmful to the control transactions that have accessed to it. Note that in this definition, we include the difference of $\mathcal{V}_i^{max}$ and $\mathcal{V}_i^{min}$ as a weighting factor.

Following the definition in Eq. 8.1, we define the *quality of control (*QoC*)* obtained from completing a control job. In Eq. 8.2, we use $\mathcal{Q}_i^c(t)$ to denote the *QoC* obtained from completing a job of τ_i^c as a function of time. Since the impact of a stale data object on a control transaction is application dependent, in here, we assume that $\mathcal{Q}_i^c(t)$ is the sum of the *QoD* $\mathcal{Q}_k^u(t)$ ($\tau_k^u \in \Omega_i$) at the time when a job of τ_i^c is finished, weighted by the importance

factor $\alpha_{i,k}$, which indicates the relative importance of a data object to a control transaction.

$$Q_i^c(t) = \sum_{k=1}^{|\Omega_i|} \alpha_{i,k} \cdot Q_k^u(t) \tag{8.2}$$

This definition is suitable for the control transactions which access to a set of data objects to derive a result from the set of data objects, *e.g.*, a control job calculates the mean pressure of a set of neighboring pressure sensors in controlling the movement of a grasping hand. If a control job misses its deadline, the *QoC* returned from it will be reduced to zero.

8.2 Co-Scheduling Algorithm *Co-LALF*

Although the scheduling algorithms presented in previous chapters, *e.g.*, *DS-FP* and *DS-LALF*, are effective in guaranteeing the validity of all the real-time data while minimizing the CPU workload imposed by the update transactions, it ignores the performance of the control transactions by assigning them to lower priories compared with the update transactions. In the update and control co-scheduling problem, the performance goal is to find a schedule for the given sets of control and update transactions such that all the control transactions can meet their deadlines and the *QoD* of data objects observed by them is maximized. For solving the co-scheduling problem, in this section, we first describe the baseline algorithms and discuss their limitations. Then, we extend *DS-LALF* to be a co-scheduling algorithm called *Co-LALF* by taking control transactions into consideration in defining the generation period for the update transactions and scheduling them.

8.2.1 Baseline Co-Scheduling Algorithms and Their Limitations

In *ML*, *DS-FP*, *DS-EDF* and *DS-LALF*, the control transactions are always assigned lower priorities in scheduling compared with the update transactions. We call this method in the co-scheduling as the *Update First* (*UF*) method. *UF* can maximize the *QoD* but the schedulability of the control transactions can be greatly affected especially if the update

workload is heavy. Another alternative for the co-scheduling is to assign higher priorities to the control transactions compared with the update transactions. We call this method as the *Control First* (*CF*) method. The tradeoff of *CF* is that it may maximize the schedulability of the control transactions but some of the update transactions may not be able to meet their deadlines, and consequently, the *QoD* of data objects can be seriously degraded.

In addition to the relative priorities between the update and control transactions, in the co-scheduling problem, we also need to consider how to schedule the jobs within each group of transactions and how to generate update jobs from an update transaction. For example, we may combine *ML* with *UF* and *CF* to determine the periods and deadlines of the update transactions using a fixed validity interval, and then apply *Deadline Monotonic (*DM*)* scheduling [95] to schedule the transactions. The control transactions can also be scheduled using *DM* according to their relative deadlines. However, interleaving the execution of the update and control jobs together can make some of them miss the deadlines and the overall performance of the system will become unpredictable. This is unacceptable in real-time sensing and control systems.

Notice that in *CF*, the calculation of the update transaction period is based on its worst-case response time without considering the preemptions from any control transactions. To minimize the impacts of the control transactions on the scheduling of the update transactions in *CF*, we may include the preemptions from the control transactions in deriving the worst-case response time for an update transaction. We call this method as the *Control First with Preemption* (*AF-P*). The tradeoff is that the calculated periods for the update transactions using *AF-P* will be shorter compared with *CF*, and the imposed update workload will be higher.

8.2.2 The *Co-LALF* Algorithm

To overcome the aforementioned drawbacks of *UF* and *CF*, it is important to schedule the update and control jobs dynamically. In this section, we present the *Least Actual Laxity*

First co-scheduling algorithm (*Co-LALF*) which is an extension of the *DS-LALF* algorithm and aims at providing better schedulability and overall performance in real-time sensing and control systems.

The Principles of *Co-LALF*

Similar to the *DS-LALF* algorithm, *Co-LALF* defines the actual laxity of an update job to be the number of idle slots between its previous job's deadline and its current deadline. As an extension, *Co-LALF* further defines the actual laxity of a control job to be the number of idle slots between its release time and deadline. With this definition, *Co-LALF* can schedule both update jobs and control jobs together using the consistent criteria to select the next job for execution. Another important extension of *Co-LALF* is that instead of using a single data validity interval, a range of validity intervals is adopted to define the validity constraints for real-time data objects as introduced in Section 8.1.2. The benefit of this extension is to provide a tradeoff between quality of data and the system schedulability especially for the control transactions. When a control job is not schedulable, a schedule adjustment mechanism will be applied to reallocate the time slots previously allocated to certain update jobs to the control job to meet its deadline. The validity intervals of the involved update transactions will be increased accordingly as long as they do not exceed their maximum validity intervals.

The Algorithm

Alg. 25 presents the details of the *Least Actual Laxity First* co-scheduling algorithm, *Co-LALF*. In the algorithm, we use $\Theta^u(a,b)$ ($\Theta^c(a,b)$) to denote the number of time slots which have already been allocated to high-priority update (control) jobs in time interval $[a,b)$, and $\Theta(a,b) = \Theta^u(a,b) + \Theta^c(a,b)$. We use $rc_{i,k}$ and $n_{i,k}$ to denote job $J_{i,k}$'s remaining execution time and the actual laxity of $J_{i,k}$, respectively. If $J_{i,k}$ is a control job, then $n_{i,k}$ is the actual laxity from its release time $r_{i,k}$ to its absolute deadline $d_{i,k}$ minus $rc_{i,k}$, *i.e.*,

Alg 25 Co-Scheduling Algorithm of *Co-LALF*

Input: Update transaction set $\mathcal{T}^u$ and control transaction set $\mathcal{T}^c$.
Output: A partial schedule $\mathcal{S}_{LALF}$ if $\mathcal{T}^u$ and $\mathcal{T}^c$ are schedulable. Otherwise returns failure.

```
1: D_max = max_i{V_i^min};
2: Enqueue all the first update job (J_{i,0}^u) with d_{i,0}^u = V_i^min.
3: Enqueue all control jobs (J_{i,j}^c) with d_{i,j}^c ≤ D_max.
4: CalcIdleSlot(D_max);
5: while TRUE do
6:    if the highest-priority job J_{i,k} in Q_LALF ∈ T^c then
7:       if n_{i,k} ≥ 0 then
8:          Allocate the earliest idle time slot from r_{i,k} for J_{i,k};
9:          rc_{i,k} − −;
10:         if rc_{i,k} == 0 then
11:            Enqueue J_{i,k+1} in Q_LALF;
12:            if d_{i,k+1} > D_max then
13:               D_max = d_{i,k+1};
14:            end if
15:         end if
16:      else
17:         IsAdjustable = ScheduleAdjustment(J_{i,k});
18:         if IsAdjustable == FALSE then
19:            Return Failure;
20:         end if
21:      end if
22:   else
23:      if n_{i,k} ≥ 0 then
24:         allocate the first available idle time slot t backward from d_{i,k} for J_{i,k};
25:         rc_{i,k} − −;
26:         if J_{i,k} is complete then
27:            r_{i,k} = t − δ_i.
28:            d_{i,k+1} = r_{i,k} + V_i^min;
29:            Enqueue job J_{i,k+1} into Q_LALF;
30:            if d_{i,k+1} > D_max then
31:               D_max = d_{i,k+1};
32:            end if
33:         end if
34:      else
35:         d_{i,k} = d_{i,k} + 1;
36:         if d_{i,k} > r_{i,k−1} + V_i^max then
37:            Return Failure;
38:         end if
39:      end if
40:   end if
41:   Enqueue all control jobs (J_{i,j}^c) with d_{i,j}^c ≤ D_max.
42:   CalcIdleSlot(D_max);
43: end while
```

Alg 26 ScheduleAdjustment

Input: control job $J^c_{i,k}$ with $n_{i,k} < 0$.
Output: A feasible schedule for $J^c_{i,k}$.

```
1: if Θ^u(r^c_{i,k}, d^c_{i,k}) < rc^c_{i,k} then
2:    Return FALSE;
3: else
4:    n_{p,q} = −∞;
5:    for each update job J^u_{a,b} with execution in [r^c_{i,k}, d^c_{i,k}) do
6:       if n_{a,b} > n_{p,q} || (n_{a,b} == n_{p,q} && V_a − V^min_a < V_p − V^min_p) then
7:          n_{p,q} = n_{a,b};
8:          J_{p,q} = J^c_{a,b};
9:       end if
10:   end for
11:   Assign the earliest time slot of J_{p,q} in [r^c_{i,k}, d^c_{i,k}) to J^c_{i,k};
12:   Increment J_{p,q}'s remaining execution time;
13:   rc^c_{i,k} − −;
14:   Return TRUE;
15: end if
```

$n_{i,k} = d_{i,k} - r_{i,k} - \Theta(r_{i,k}, d_{i,k}) - rc_{i,k}$; If $J_{i,k}$ is an update job, $n_{i,k}$ is the actual laxity between $d_{i,k-1}$, the deadline of $J_{i,k-1}$, and $d_{i,k}$, *i.e.*, $n_{i,k} = d_{i,k} - d_{i,k-1} - \Theta(d_{i,k-1}, d_{i,k}) - rc_{i,k}$. Note that $n_{i,k}$ may be negative. The algorithm maintains a job queue Q_{LALF} for the pending update and control jobs. In each step of the algorithm, we select the job in Q_{LALF} with the least actual laxity for executing one time slot. The calculation of the actual laxity for a job is summarized in Alg. 27.

As shown in Alg. 25, in the initialization phase, we first put the first job $J^u_{i,0}$ of each update transaction τ^u_i into Q_{LALF} and set its deadline as $\mathcal{V}^{min}_i$. Then, we set D_{max} to be $\max_i\{\mathcal{V}^{min}_i\}$ and enqueue all the control jobs whose deadlines are not larger than D_{max}. We invoke the function **CalcIdleSlot()** to calculate the actual laxity for each job in Q_{LALF} and select the job with the least actual laxity for execution. If more than one job have the same number of idle time slots, we select the job with the earliest deadline.

If the highest-priority job $J_{i,k}$ is a control job and $n_{i,k} \geq 0$, $J_{i,k}$ will be allocated the first available idle time slot from $r_{i,k}$ and its remaining execution time $rc_{i,k}$ will be decremented by one. If the actual laxity is not sufficient to finish $J_{i,k}$ to make $J_{i,k}$ schedulable,

Alg 27 CalcIdleSlot

Input: deadline t_d.
Output: $n_{i,k}$ for each job $J_{i,k}$ in Q_{LALF} with deadline $d_{i,k} \leq t_d$.

```
1: for each job J_{i,k} with d_{i,k} <= t_d do
2:     n_{i,k} = d_{i,k} − r_{i,k} − rc_{i,k} + 1;
3:     for t = r_{i,k} to d_{i,k} do
4:         if time slot t is allocated to a higher-priority job then
5:             n_{i,k}−−;
6:         end if
7:     end for
8: end for
```

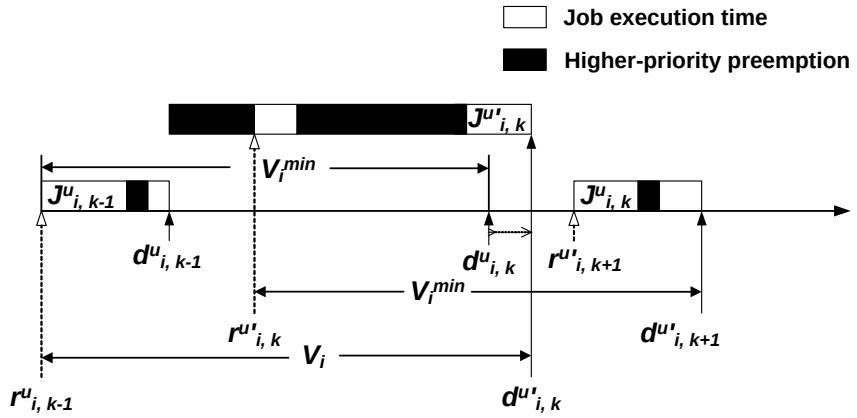

Figure 8.2: Increasing the validity from $\mathcal{V}_i^{min}$ to a larger $\mathcal{V}_i$ for update job $J^u_{i,k}$

function **ScheduleAdjustment()** (Alg. 26) is invoked to adjust the schedule in $[r_{i,k}, d_{i,k})$ so that $J_{i,k}$ can be finished before its deadline $d_{i,k}$. The schedule adjustment reallocates the time slots previously allocated to certain update jobs to the control job $J_{i,k}$ and the remaining execution time of the update jobs will be increased accordingly. Then, it selects the update job with the larger actual laxity. If more than one update jobs have the actual laxities equal to zero, it will select the one with the smallest increase in validity interval compared with its minimum validity interval. If Alg. 26 fails to adjust the schedule, the feasible schedule $\mathcal{S}_{LALF}$ cannot be found and Alg. 25 will return failure to indicate that the sets of update and control transactions are non-schedulable.

If the highest-priority job $J_{i,k}$ is an update job and $n_{i,k} \geq 0$, $J_{i,k}$ will be allocated the first available idle time slot, say time t, backward from $d_{i,k}$ and its remaining execution time $rc_{i,k}$ will be decremented by one. If $rc_{i,k}$ reaches 0 after this allocation at time t, the release time $r_{i,k}$ of $J_{i,k}$ will be set as $t - \delta_i$. The next job of τ_i^u, $J_{i,k+1}$, will be put into Q_{LALF} with deadline $d_{i,k+1}$ set as $r_{i,k} + \mathcal{V}_i^{min}$. If $n_{i,k} < 0$, then the deadline $d_{i,k}$ of $J_{i,k}$ will be increased by one as long as the deadline is no larger than the maximum deadline allowed by the system, *i.e.*, $r_{i,k-1} + \mathcal{V}_i^{max}$. If the remaining execution time of $J_{i,k}$ is still larger than zero when the deadline $d_{i,k}$ reaches $r_{i,k-1} + \mathcal{V}_i^{max}$, the transaction sets $\mathcal{T}^c$ and $\mathcal{T}^u$ are not schedulable, and the algorithm will return failure. For example, as shown in Figure 8.2, if the deadline $d_{i,k}^u$ of $J_{i,k}^u$ is set to be $r_{i,k-1}^u + \mathcal{V}_i^{min}$, $J_{i,k}^u$ is not schedulable as the derived release time $r_{i,k}^u$ is smaller than the deadline $d_{i,k-1}^u$ of $J_{i,k-1}^u$ due to preemptions from higher-priority jobs (both update and control jobs). To solve the problem, the deadline $d_{i,k}^u$ of $J_{i,k}^u$ is increased from $r_{i,k-1}^u + \mathcal{V}_i^{min}$ up to $r_{i,k-1}^u + \mathcal{V}_i^{max}$ such that $J_{i,k}^u$ is schedulable. As shown in Figure 8.2, the new deadline and release time for $J_{i,k}^u$ are $d_{i,k}^{u'}$ and $r_{i,k}^{u'}$, respectively, after extending the value for $\mathcal{V}_i$.

8.3 Performance Evaluation

In this section, we report the important results obtained from our performance studies on *Co-LALF*. We study the performance of *Co-LALF* compared with the baseline methods *UF* and *CF* to illustrate the benefits of the adaptive priority scheduling used in *Co-LALF*. In the experiments, a wide range of workloads are used to test their performance by changing the number of update transactions as well as other important parameters such as the number of control transactions and the jitter for update transactions. In addition, the default settings for the base parameters, e.g., the worst-case execution of an update transaction and number of update transactions are chosen based on the settings used in previous studies such as [94, 95] so that our results can be compared with the findings from them.

Parameter	Value	Parameter	Value
# of Update Trans.	80	# of Control Trans.	20
C_i^u	[5,10]	C_i^c	[5, 15]
$\mathcal{V}_i^{min}$	[600, 2600]	P_i^c	[200, 800]
$\mathcal{V}_i^{max}$	$2 \times \mathcal{V}_i^{min}$	D_i^c	$0.5 \times P_i^c$

Table 8.2: Parameter settings in the experiments

8.3.1 Simulation Model

The simulation model is developed according to the system model introduced in Section 8.1. It consists of a real-time controller and a fixed set of sensor nodes. Each sensor node runs an update transaction to generate update jobs following the release times determined by the adopted method, i.e., *DS-EDF* or *DS-LALF*. At the same time, a set of control transactions is maintained at the controller to generate control jobs following the pre-defined periods to access to the data objects in the real-time database. In the simulation model, we do not consider the concurrency control between the update transactions and control transactions as the conflicts can easily be resolved by the application of the concept of data similarity [47]. It is also assumed that the synchronization delay in accessing shared data object is small and can be ignored as all the data objects reside in main memory and the problem of priority inversion can be resolved by the use of the priority inheritance method [78].

8.3.2 Performance Evaluation on *Co-LALF*

In these set of experiments, we evaluate the performance of *Co-LALF* as compared with the two baseline methods *CF* and *UF*. Note that the validity interval for a data object in *CF* and *UF* is always set to be the minimum validity interval of the data object. Table 8.3.2 summarizes the baseline parameter settings used in these set of experiments.

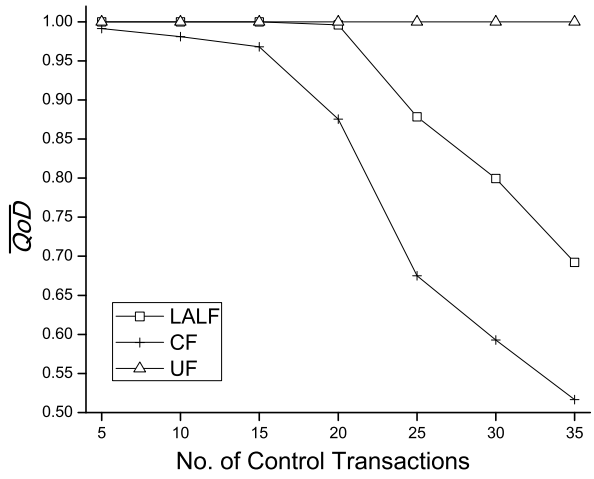

Figure 8.3: $\overline{QoD}$ Vs. No. of control trans.

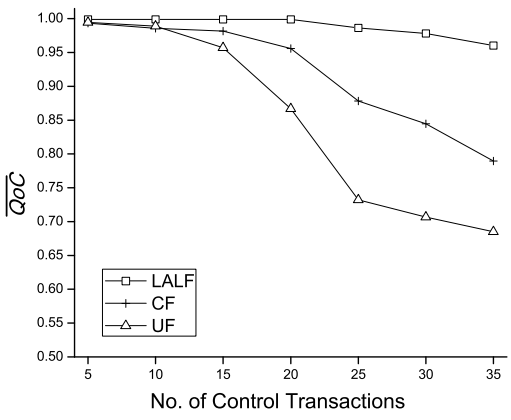

Figure 8.4: $\overline{QoC}$ Vs. No. of control trans.

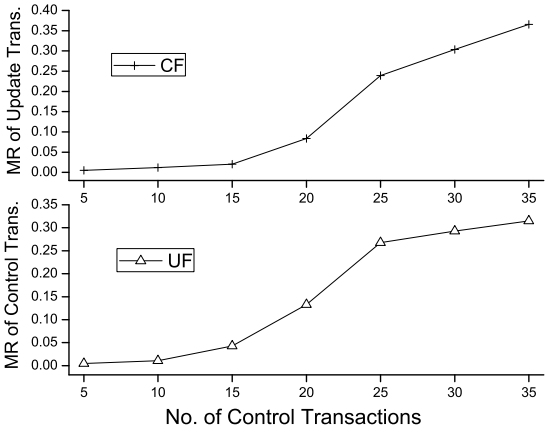

Figure 8.5: Miss rates

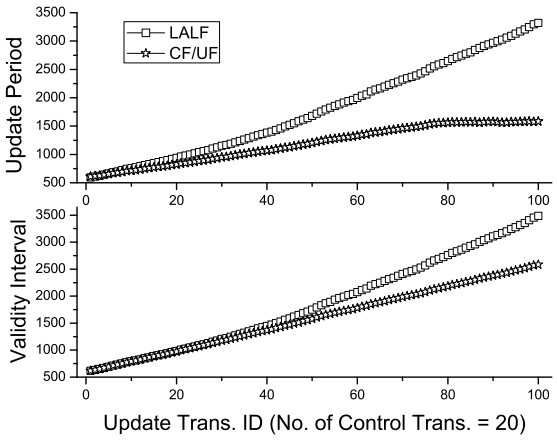

Figure 8.6: Update period / Validity interval

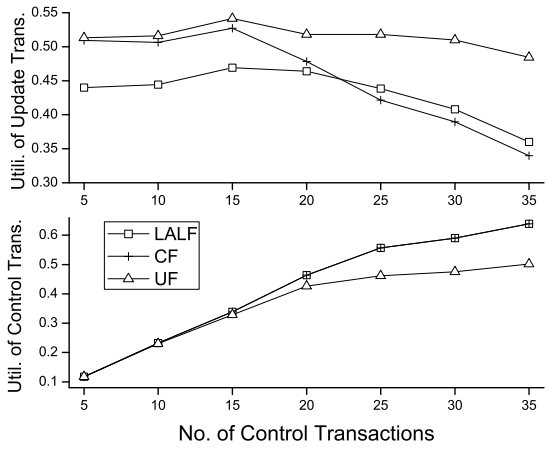

Figure 8.7: Workload vs. No. of control trans.

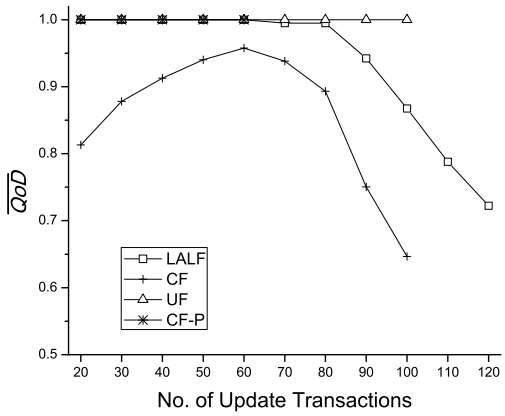

Figure 8.8: $\overline{QoD}$ Vs. No. of update trans.

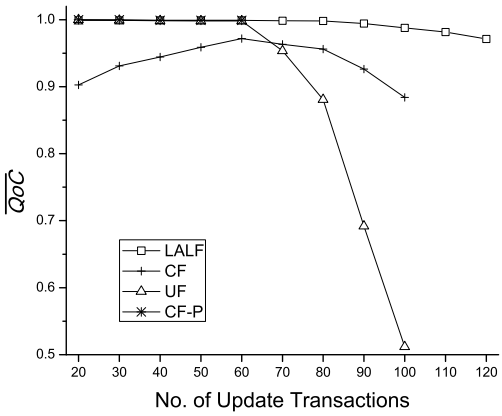

Figure 8.9: $\overline{QoC}$ Vs. No. of update trans.

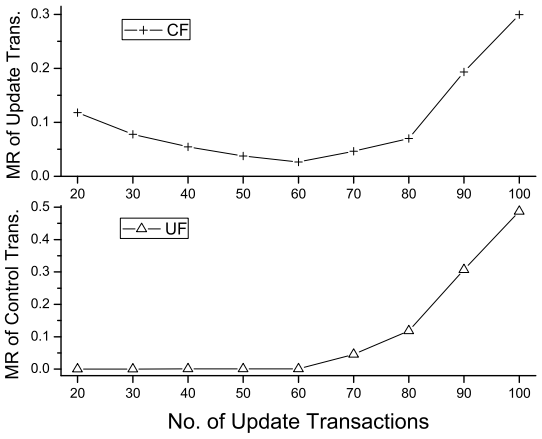

Figure 8.10: Miss rates

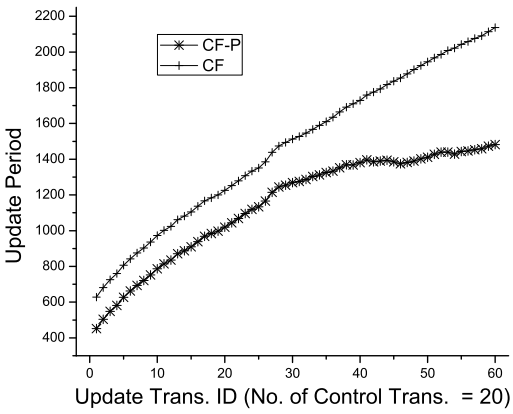

Figure 8.11: Update periods

Impacts of Control Transaction Workload

In this set of experiments, we first fix the number of update transactions at 100 and vary the control transaction workload in the systems by increasing the number of control transactions from 5 to 35 (when the total workload is close to 1). Figure 8.3 depicts the average QoD of all the data objects ($\overline{QoD}$) obtained from *Co-LALF*, *CF* and *UF*. As shown in Figure 8.3, although *UF* gives better $\overline{QoD}$ compared with *CF* and *DS-LALF*, the average *QoC* of all the control transactions ($\overline{QoC}$) in *UF* decreases dramatically along with an increase in control transaction workload as shown in Figure 8.4. This is because as shown in Figure 8.5, the miss rate of the control transactions (defined as the number of deadline missed control jobs over the total number of control jobs generated) in *UF* increases rapidly with an increase in control transaction workload as they are executed at lower priorities compared with the update transactions. Note that missing the deadline of a control job will reduce its *QoC* to zero. Similarly, $\overline{QoC}$ of *CF* also decreases with an increase in control transaction workload as a result of poor $\overline{QoD}$ (Figure 8.3) due to the high miss rate of the update transactions (Figure 8.5). Compared with *UF* and *CF*, *Co-LALF* achieves better $\overline{QoC}$ (close to 1) as shown in Figure 8.4 as all the control jobs can be completed before the deadlines

and at the same time *Co-LALF* can maintain a high $\overline{QoD}$. The lower $\overline{QoD}$ of *Co-LALF* as compared with *UF* when the control transaction workload is heavy (i.e., number of control transactions is more than 20) is because in cases the deadlines of some control jobs cannot be satisfied, the validity intervals of some data objects are extended to meet the deadlines of the control jobs. As shown in Figure 8.6, the validity intervals and generation periods of the update transactions in *Co-LALF* are consistently longer than that in *UF* and *CF* especially for lower priority update transactions (the transactions with larger transaction IDs). This is because for lower priority update transactions, the preemptions from higher priority jobs are longer. Thus, larger validity intervals and longer periods are required to meet the deadlines of the control jobs.

As shown in Figure 8.7, the update transaction workload in *UF* is the highest while *CF* and *Co-LALF* give higher control transaction workload. The drop in update transaction workload in *CF* with an increase in control transaction workload is due to the high miss rate of the update transactions. Similarly, the lower control transaction workload in *UF* is due to high miss rate of the control transactions. It is important to note that the deadlines of all the control jobs are satisfied in *Co-LALF*, and the update transaction workload of *Co-LALF* is lower than both *CF* and *UF* when the control transaction workload is not heavy, i.e., smaller than 22 control transactions, as it uses the aperiodic update model by extending the periods for update job generation. When the control transaction workload is heavy, the update transaction workload of *CF* is lower than that of *Co-LALF* as many update jobs miss deadlines.

Impact of Update Transaction Workload

In this set of experiments, we fix the number of control transactions at 20 and vary the update transaction workload. In the experiments, we also include *AF-P* for comparison. Consistent with the results presented in the previous set of experiments, although *UF* gives good values of $\overline{QoD}$ (Figure 8.8) and $\overline{QoC}$ (Figure 8.9) when the update transaction work-

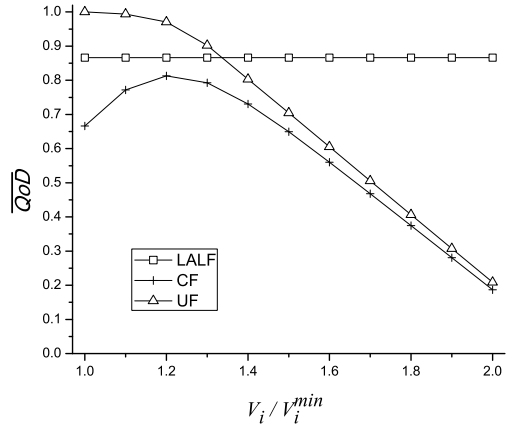

Figure 8.12: $\overline{QoD}$ Vs. $\mathcal{V}_i/\mathcal{V}_i^{min}$

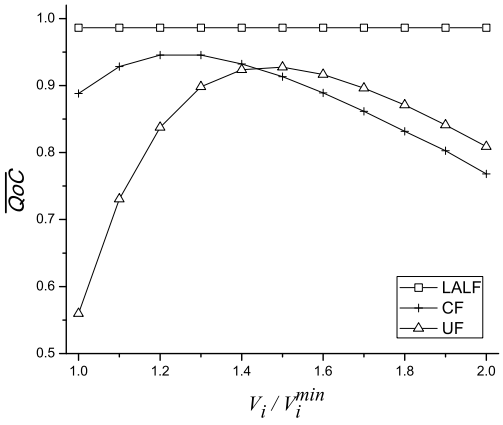

Figure 8.13: $\overline{QoC}$ Vs. $\mathcal{V}_i/\mathcal{V}_i^{min}$

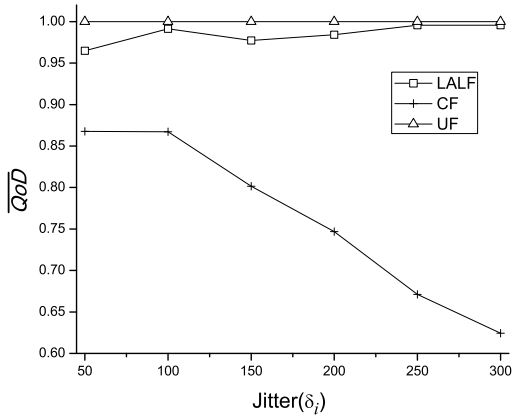

Figure 8.14: $\overline{QoD}$ Vs. Jitter

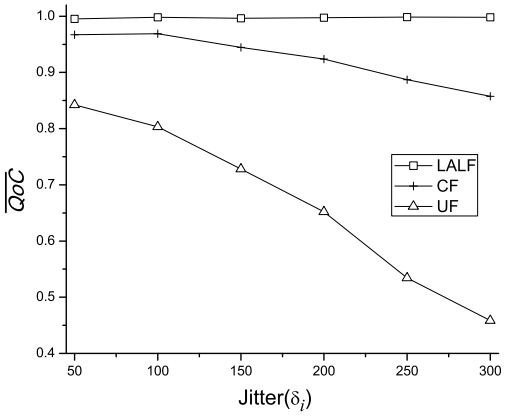

Figure 8.15: $\overline{QoC}$ Vs. Jitter

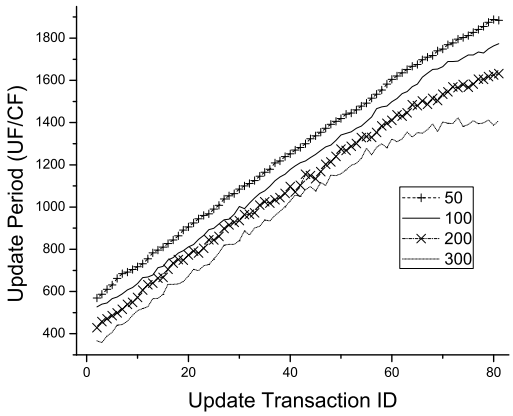

Figure 8.16: Update periods (CF/UF)

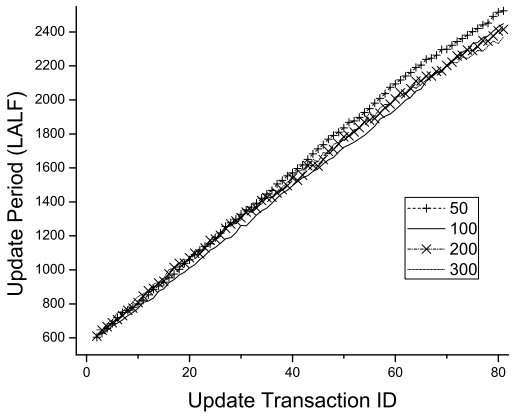

Figure 8.17: Update periods (LALF)

load is low, $\overline{QoC}$ of *UF* is much lower compared with *CF*, *AF-P* and *Co-LALF* when the update transaction workload is heavy (Figure 8.9). Again, it is due to high miss rate of the control transactions when the update workload is heavy. On the other hand, as shown in Figure 8.9, $\overline{QoC}$ of *CF* is consistently lower than that of *Co-LALF* and *AF-P* due to higher miss rate of the update transactions and lower $\overline{QoD}$. As shown in Figure 8.9, although both *AF-P* and *Co-LALF* give the best $\overline{QoC}$ with values close to 1 when the update transaction workload is low, the set of update transactions in *AF-P* become non-schedulable when the number of control transactions is more than 60. This is because in *AF-P*, when calculating the periods and deadlines for the update transactions, the preemptions from the control transactions are included in deriving their worst case response times. When the number of control transactions is large, the preemptions from them will be very long. This will make the periods and deadlines of the update transactions very small (Figure 8.11) and make them non-schedulable. Unlike *AF-P*, in *Co-LALF*, the $\overline{QoD}$ and $\overline{QoC}$ remain close to 1 even when the update transaction workload is heavy. Another important observation from the figures is that both *UF* and *CF* cannot provide a schedule for their update transactions when the update transaction workload is greater than 100 tasks. However, *Co-LALF* can

still schedule them as its update transaction workload is lower by the use of the aperiodic model for update job generations.

Different Validity Intervals for UF and CF

In this set of experiments, we fix the number of update transactions and control transactions at 20 and 80 respectively and increase the length of the validity interval $\mathcal{V}_i$ from $\mathcal{V}_i^{min}$ to $\mathcal{V}_i^{max}$ to reduce the update workload and its impacts on the scheduling of the control transactions in *UF* and *CF*. As shown in Figure 8.12, although $\overline{QoD}$ of *UF* decreases consistently with an increase in $\mathcal{V}_i/\mathcal{V}_i^{min}$ (the ratio of the validity interval for each update transaction and its $\mathcal{V}_i^{min}$), its $\overline{QoC}$ increases gradually when $\mathcal{V}_i/\mathcal{V}_i^{min}$ (Figure 8.13) is small as a result of lower update transaction workload and lower miss rate of the control transactions. However, $\overline{QoC}$ of *UF* decreases consistently with a further increase in $\mathcal{V}_i/\mathcal{V}_i^{min}$ as a result of lower $\overline{QoD}$. The impacts of $\mathcal{V}_i/\mathcal{V}_i^{min}$ on *CF* are similar. Lower $\overline{QoD}$ and $\overline{QoC}$ of *CF* as shown in Figure 8.12 and Figure 8.13 are due to larger update periods and lower $\overline{QoD}$, respectively. An important observation from Figure 8.13 is that even *CF* and *UF* use larger values of $\mathcal{V}_i/\mathcal{V}_i^{min}$, $\overline{QoC}$ of *Co-LALF* is still the best and all control jobs can be completed before their deadlines in *Co-LALF*.

Impacts of δ_i of Update Transactions

In this set of experiments, we fix the number of update transactions and control transactions at 20 and 80 respectively and vary the value of δ_i to evaluate the impacts of jitters of update job transmissions on the system performance. As shown in Figure 8.15, $\overline{QoC}$ of *Co-LALF* is consistently better than that of *CF* and *UF* and it is less affected by the increase in value of δ_i. This is because an increase in value of δ_i (from 50 to 300) significantly reduces the periods of the update transactions in *UF* and *CF* as shown in Figure 8.16. However, the periods of the update transactions in *Co-LALF* remain similar under different values of δ_i (Figure 8.17). $\overline{QoC}$ of both *UF* and *CF* decreases when the value of δ_i increases as a result

of high miss rate of the control transactions in *UF* and poor $\overline{QoD}$ in *CF*.

8.4 Summary

In this chapter, we extend *DS-LALF* to be a co-scheduling algorithm for cyber-physical sensing and control systems where meeting the deadlines of the control transactions and maintaining the temporal validity of real-time data objects are two important issues to ensure the effectiveness in event monitoring. We propose a novel dynamic co-scheduling algorithm called *Co-LALF* whose performance goal is to obtain a schedule such that the deadline constraints of all the control transactions can be satisfied while the QoD are maximized. As shown in the performance studies, the dynamic scheduling policies in *Co-LALF* can effectively improve the system performance by providing a better QoD and meeting the deadlines of all the control transactions.

Chapter 9

CPS Application: A Cyberphysical Avatar

Based on the networking infrastructure and data management techniques proposed in this thesis, many interesting and important cyber-physical systems can be built to provide secure, reliable and real-time services and support a wide range of mission-critical applications. In this chapter, we will present one of the CPS applications that we are building called cyberphysical avatar. [1] The cyberphysical avatar project targets at building a semi-autonomous robotic system that can adjust to an unstructured environment and perform physical tasks subject to critical timing constraints while under human supervision.

Despite the impressive progress by the robotics community in recent years, we are still quite a way from being able to trust fully autonomous robots to carry out mission-critical and safety-critical operations by themselves. On the other hand, today's unintelligent teleoperated devices cannot be counted on to perform well in physically difficult and unstructured environments. Short of a gigantic leap in technology that creates intelligent fully autonomous robots capable of functioning in an unstructured environment, we propose to chart a pathway to evolve the capability of teleoperated robotic devices from primitive

[1] We thank Prof.Luis Sentis and Prof. Risto Miikkulainen for their contributions to this joint research project.

mechanical remote-control to trustable autonomy and more intelligent teleoperation.

Rather than trying to build fully autonomous robots from scratch, in the cyberphysical avatar project, we do it gradually through less and less human teleoperation, all the while deploying these robots in actual tasks. This is a new and different approach, and likely to result in practical applications and advances much sooner, and in more robust and better adapted autonomous robots in the end.

A cyberphysical avatar is semi-autonomous in that there are actions it must take without human intervention because of the relatively short timing constraints, *e.g.*, the control loop that maintains a fast walking gait of the robot. On the other hand, a cyberphysical avatar should not be programmed to deal with only a fixed set of scenarios because we cannot foresee all the contingencies in all operational environments, e.g., a building on fire in a rescue mission. An effective interface between the cyberphysical avatar and its human supervisor is essential for success, and this requires the cyberphysical avatar to be designed for predictable and timely response. As the cyberphysical avatar gains more physical skills, it can be trusted to perform more subtasks on its own.

There are many technical challenges in realizing the cyberphysical avatar concept. These challenges can be categorized into three topics below:

- **Dynamics and control of humanoid avatars:** The cyberphysical avatar must be able to perform the physical tasks in an unstructured and uncertain environment. In this area, we need to develop a methodology for modeling the dynamic behavior of physical avatars interacting with unstructured environments and controllers that can adapt to the changing physical conditions. We need to develop software foundations that encapsulate the physical skills supporting the full range of teleoperated to autonomous behaviors.

- **Evolutionary learning of skills under environment and performance constraints:** Evolutionary approach is needed to learn the continuous control parameters of the skills, as well as their discrete composition. The cyberphysical avatar must be able

to acquire skills so that it can perform time-critical tasks autonomously. The level of autonomy and the criticality of the timing constraints that can be satisfied for practical physical tasks requires advances in learning theory and engineering validation. Moreover, learning strategies need to be applied to learn the tradeoff between teleoperation and autonomy.

- **Supporting reliable, real-time avatar-human communication:** The cyberphysical avatar must be able to operate untethered and maintain timely and reliably communication with the controller that is hierarchically implemented with the human supervisor at the top of the control hierarchy. The combination of real-time and robust communication in a wireless environment where communication paths may be disrupted requires advance in both algorithm design and engineering validation. We need a switching policy between teleoperated and autonomous behaviors that is based on communication quality as the primary metric for making switching decisions.

The rest of this chapter will describe in more detail the system architecture of the cyberphysical avatar and the progress we have made in formalizing and partially solving the above technical challenges. The cyberphysical avatar is fully operational at this point; but we envision much more new research and new ideas to be pursued in order to perfect the collaboration between the cyberphysical avatar and human supervisor.

9.1 Control of Wheeled Humanoid Avatars in Unstructured Environments

The cyberphysical avatar must be able to maneuver in irregular terrains while performing accurate physical whole-body compliant interactions with the environment and with human operators. To attain these capabilities, skill modeling and control in unstructured environments must be carefully designed. In this section, we first describe the dynamic model of the wheeled base of our Dreamer/Meka humanoid robot under varying contact conditions.

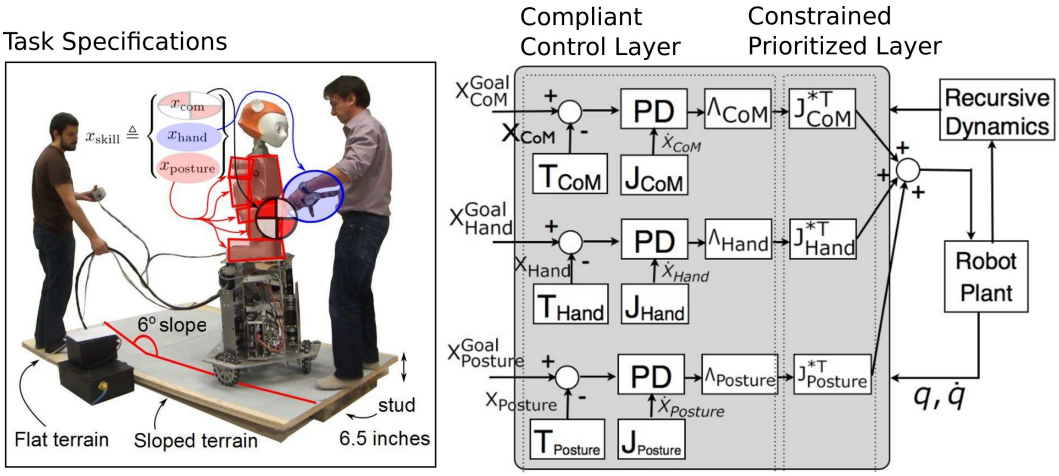

Figure 9.1: Whole-body compliant control with prioritized tasks. Left hand side of figure shows the Dreamer crosses a terrain with a slope while responding to human interaction. And right hand side of the figure shows closed loop dynamic controller producing joint torque outputs based on Center of mass, hand position and posture with prioritized Jacobians.

We then present our model of the whole-body compliant skill of the robot and the hierarchical control structures that is used to handle task conflicts during the execution of the behavior.

9.1.1 Dynamic Model of the Wheeled Base

In unstructured and uncertain environments, wheel-based avatars will often be in a situation of marginal contact, i.e. not all the wheels are in contact. As such, the dynamics of the robot, need to represent the contact state of the robot, its effect on center of mass balance and the conservation of angular and linear momentum due to marginally-stable contact conditions. This characteristics become even more critical when the robot engages into manipulation tasks while maintaining marginal contacts.

We exploit the model of the robot under varying contacts by leveraging the generalized contact consistent Jacobian developed in [74] which specifies that for a given contact state C_m the generalized Jacobian of an operational task (e.g. one of the robot's hands) is equal to

$$J^*_{\text{task},C_m} \triangleq J_{\text{task}}\ \overline{UN}_{C_m}, \tag{9.1}$$

where J_{task} is the Jacobian of the hand Cartesian point with respect to an inertial frame outside of the mobile base, U describes the underactuated (i.e. uncontrollable directions) of the base due to the contact state, N_{C_m} describes the current contact state (i.e. how many wheels are in contact), and the operator $\overline{(.)}$ indicates a dynamically consistent generalized inverse of the argument. Therefore the control of the operational task (e.g. the control of the robot's hand) while taking into account the mobility of the base and the uncertain contact state is equal to

$$\Gamma_{C_m} = J^{*T}_{\text{task},C_m}\ F_{\text{task}} \tag{9.2}$$

where F_{task} is the force or impedance command to control the hand, J^*_{task,C_m} is the whole-body task Jacobian including the base contact state (i.e. how many wheels are in stable contact), and Γ_{C_m} is the whole-body command of torques sent to the base and upper humanoid torso motors.

9.1.2 Skill Definition and Hierarchical Control Structure

In whole-body compliant control (WBC), a task is defined via a mapping between the robot's N-dimensional joint configuration and some M-dimensional space which describes an objective that the controller should achieve. The skill is defined as a juxtaposition of multiple operational tasks to help translate between high-level goals (such as provided by planning algorithms) and the operational tasks. In our environment, a skill is a human readable file (e.g. YAML) describing the points or coordinates of the robot that are to be simultaneously controlled to accomplish a behavior, plus their respective control policies, and plus their hierarchical priorities in the execution pipeline. In this work, however, as to be elaborated in Section 9.2, the control policies of the skill will be learned through machine

learning approaches.

Having now many operational task processes to simultaneously optimize, as defined in the skill, either as force or impedance processes, we propose to use the following control structure in Eq. 9.3. The intuition behind this control structure is to instantiate several tasks, each of which tries to drive the robot toward some state. The task contributions are accumulated using null space projections to ensure that lower-priority tasks do not interfere with higher levels. The motion is thus determined by each task in combination with their priorities. This structuring provides two orthogonal ways of changing robot behavior, either by influencing the tasks (e.g. changing their gains or goals) or by rearranging the hierarchy (e.g. inserting tasks or locally inverting their ordering).

$$\Gamma_{C_m} = \sum_k \left(J^{*T}_{k|\text{prec}(k),C_m} F_k \right) + N^{*T}_{t,C_m} \Gamma_{\text{posture}} + J^{*T}_{i|l,C_m} F_{\text{int}}, \tag{9.3}$$

In Eq. 9.3, F_k is the task space force or impedance command for the k-th operational task, $J^{*}_{k|\text{prec}(k),C_m}$ is the prioritized contact consistent Jacobian of the task, Γ_{posture} is the command to optimize the posture behavior, F_{int} is the command to optimize the internal forces between the arms and the mobile base, and $J^{*}_{i|l,C_m}$ is the Jacobian of the internal forces. This structure is a derivation of our previous work on whole-body compliant control found in [75].

In the control structure, the low-level tasks describing the skill are aggregated using a hierarchy, where more relevant tasks, such as those who ensure the fulfillment of physical constraints appear first, while those dealing with the operational behavior appear with less priority. Fig. 9.1 gives an example where the skill is composed of three tasks. The first task is maintaining coordinates of center of mass(CoM) to prevent the robot from falling down on irregular terrain. Second task is compliant hand position which enables the robot to respond compliantly to human interaction. The posture task here is utilizing remaining degree of freedom to stabilize self-motion and converge to a human-like posture. Lower-priority tasks operate in the null space of all higher priority tasks. So when the terrain changes the

CoM task will temporarily override non-critical tasks in order to prevent falling. The task becomes unfeasible when the current higher priority tasks use all the dynamic redundancy. This event can be easily monitored and used to stop the behavior and communicate the problem to a high level planner.

Because our control structures use effectively the dynamic and contact model of the physical avatar in its environment, they are able to optimize all task processes simultaneously within the contact stance, thus achieving precise tracking of forces and trajectories. Moreover, posture behavior, which is specified as an optimization criterion instead of a trajectory is also optimized within the residual manifolds left over by the priority tasks.

9.2 Skill Acquisition by Machine Learning

Although much of the operation of the robot can be based on carefully designed control algorithms, there are two issues where machine learning methods can prove crucial: (1) conversion of human operator behaviors to robot behaviors, and (2) optimization of robot behaviors. In both of these cases, it is possible to come up with measures of how good the behaviors are, but the optimal behaviors are not known. Therefore, machine learning methods based on exploration need to be used. In this section, a particularly powerful such a method, neuroevolution, is described first, followed by its application to train the learning skills of the grasper on the Dreamer humanoid robot to pick up objects with any shape.

9.2.1 Learning Robust Nonlinear Control through Neuroevolution

In the neuroevolution approach, evolutionary optimization method such as a genetic algorithm is used to construct the structure and the connection weights of a neural network so that the network performs as well as possible in a given task [60, 25]. The neural network can be recurrent, implementing a sequence memory, and thereby making it possible to use the approach to discover sequential behaviors such as robot navigation, arm control, and grasping.

Neuroevolution differs from other machine learning methods in two important ways. First, most such methods are supervised, which means that they learn behavior that approximates a given set of examples [36]. The examples need to be carefully constructed to represent correct, or optimal, performance—the learning system will then learn a function that interpolates between them smoothly. For instance, in grasping, a number of grasping situations (configuration of the hand, shape and position of the object) need to be created together with the optimal grasping behavior. In general, it is difficult to come up with such examples because optimal behavior is often not known; also, it is difficult to cover all possible situations, making the behavior incomplete. In contrast, neuroevolution learning is based on exploration and reinforcement: a population of neural networks is evolved through crossover and mutation, directed only by how well each network performs. It is thus possible to discover successful behaviors that human designers would find difficult to construct, and behaviors that are more general.

Second, other methods that are designed to learn under sparse reinforcement, such as Q-learning, or value function learning in general, assume that the current state of the system is completely known [85]. If objects are occluded, or the situation is changing, it is difficult for them to anticipate what will happen (because the observed values of actions cannot be associated with the correct state). In contrast, the neural networks employed in neuroevolution can disambiguate the state based on their sequence memory. Previous sensor values are part of the state representation, making it possible to understand how the world is changing, and how to respond to it optimally. Such an ability is particularly useful in domains such as grasping that are highly dynamic and where sensors are limited.

The neuroevolution approach can thus be used as a training mechanism for the Dreamer robot as well. In particular, it is well suited for learning skills such as picking up an object. In the following, we will first describe the physics of the grasper on the Dreamer/Meka humanoid robot and then present the training details. Our training is done following the NEAT [83] approach.

9.2.2 Physics of the Grasper

We simulate the Meka hand using GraspIt! [7], an open-source grasping simulation environment developed at Columbia University. GraspIt! provides mechanisms for simulating gravity, interactions between rigid objects, physical modeling, and joint movement for the specific purpose of developing efficient robotic grasps. Robotic components in particular are modeled as a series of degrees of freedom specifying parameters such as default rotational velocity, maximum torque, and relative position and rotation ranges. Each degree of freedom is connected to a kinematic chain and associated with a visual model in the three-dimensional environment. These individual components are combined to create an entire robotic apparatus. The definitions are stored as a series of XML files.

The Meka hand in particular is defined by one degree of freedom for each knuckle in each finger, as well as degrees of freedom for the thumb's rotator. The mechanics of this model will be modified in our studies in order to account for two phenomena. First, we wish to control the wrist, which isn't modeled explicitly by default. We will therefore add a wrist component to the Meka model supplied by GraspIt!. Second, most of the degrees of freedom in the Meka hand are not actuated. Each finger consists of three joints, which are all connected by a single rubber tendon. When the finger curls, all three knuckles curl in unison. We will therefore adjust the torques that we feed to the simulator to account for this interdependent joint behavior. A set of torques given to a single finger will conform with one another such that they are all equivalent to torques initiated by a stretching of the rubber tendon, which we see in the real robot.

9.2.3 Training the Grasper

To properly grasp with the Meka hand, first, we have to design an input and output layers of our target neural network, as well as a fitness function to allow a gradual climb toward an efficient grasp. Because our Meka unit is primarily controlled using the Whole-Body Control Framework, the network is only responsible to directly manipulate orientation and

position of the wrist. Thus, we designate an output node for each of these degrees of freedom.

Each neural network generated from NEAT receives several input data that illustrate our current state of the robot in an environment. Designing the input layer is less trivial. It is necessary to encode the entire state of the grasp in some way, which includes positions of the grasped object, as well as the object's shape. To reduce dependency on the shape of the grasped object, we encode the object's state by simply taking a depth map from the robot's Kinect sensor and assigning each depth data a unique input node in the neural network. In this way, the network is able to associate state of an arbitrary object in an arbitrary environment with a grasping strategy namely appropriate position and angle of the robot hand.The output of each network through NEAT is to predict where an object is and what is the best direction to grasp the object in the form of three-dimension hand positions and orientation.

The final stage of our design is to construct an adequate fitness function. The fitness function of a network n with respect to a corresponding object o is computed as the reciprocal of mean square error $\overline{M}$, the summation of distance between the center of robot's palm and a desired object, and also combined with the grasp quality metrics provided by GraspIt!. Let P_i be the predicted position of hand for grasping by the network, where $i \in x$, y, z coordination. Let O_i be a coordinate of the selected object after mouse click, where i is captured from the Kinect sensor inputs to get x, y, z coordination. Let q be a quality value after the execution of a single grasp, which is normalized into the range $[0, 1]$ by GraspIt!. Thus, the fitness function f is defined as follows:

$$f(n) = \frac{\beta}{\overline{M} + \alpha} + \gamma q = \frac{\beta}{\sum_{i \in x,y,z}(P_i - O_i)^2 + \alpha} + \gamma q \tag{9.4}$$

, where α, β and γ are constants.

The first term of the equation is measuring the distance between the hand and the object and the second term is proportional to the grasp quality. During the initial phases of

learning, the coordinate is arbitrarily chosen, so the robot hand will try to grasp at arbitrary position where it can not even touch the object. As a result, the second term of the fitness function will be almost always zero in the early generations. So in this stage, the first term is used to differentiate the fitness of the neural networks, which will train the networks to get closer to the target object. After some generations, when the hand can grasp the object the second term will start be effective and rank the results according to the grasp quality. And parameters α, β and γ is used to adjust the relative affect of these two terms.

So with this fitness function, the neural networks will first learn to get close to the object and then learn to grasp the object in right way.

9.2.4 Simulation Results

To explore the robot behavior in the real world, we adopt dynamic environment in the simulation. Because GraspIt! only has limited support for the hand model of Meka robot and only provides restricted hand movement, we have improved the hand model and movement functionality to make it work with our simulation environment.

Fig. 9.2 illustrates the overview of our simulation setup. The Kinect sensor is simulated within GraspIt! simulator which is used to provide a set of depth data as inputs for the neural networks. This array of depth data together with a two-dimensional coordinate representing mouse click from users are fed to the input layer of the NEAT. The output of the NEAT consists of hand position and hand orientation where they are sent back to the simulator for manipulating the robotic hand and evaluating the quality of grasp. The structure and the weights of the neural networks are automatically created with NEAT [83] through several generations.

The evolved neural networks are utilized to retain hand position and orientation, and we use these data to manipulate Meka hand in simulation environment for performing grasping and evaluating the quality of grasping. In this experiment, the input data contains 20×20 depth array nodes, 2 coordinate nodes. The coordinate node here denotes mouse

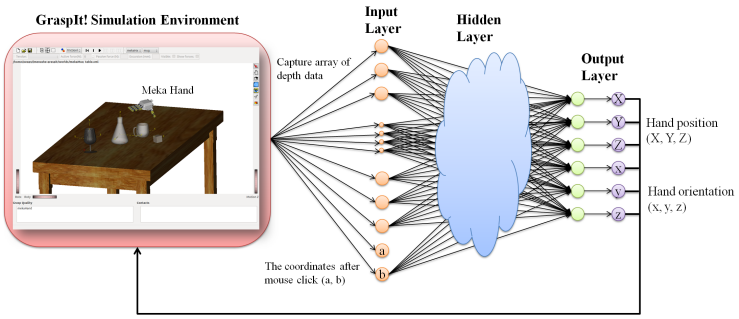

Figure 9.2: Representation of the designed grasp controller network.

click input from the user indicating the target object. So the remote human controller can click on Kinect Sensor video stream to identify the target object of the grasping. In our experiment, this coordinate is created by randomly picking a point of the target object.

These input data are directly connected to the output to form the first generation of neural networks, which are then evolved through mutation and crossover. We set the population size as 128, refine three parameters α, β and γ of the fitness function Eq. 9.4, and choose number of generations as 80. The fitness value is a function of the distance between the Meka hand and a target object plus the quality of grasping. In the fitness function, larger fitness value implies better grasping quality. As Fig. 9.3 shown, in the starting networks, the first average fitness is low because they are composed of the input data directly connected to the outputs with random weights. As the neural networks evolve through generations, performing adaptive weight and structure adjustment, the fitness value increases. After around 30 generations, it reaches around 0.8 which means the Meka hand can grasp the object more accurately with the proper position and orientation.

We also observe that after 30 generations, the fitness value oscillates around maximum value. It is possible that we do not perfectly characterize our fitness function, and more fine tune shall be made to further improve the result.

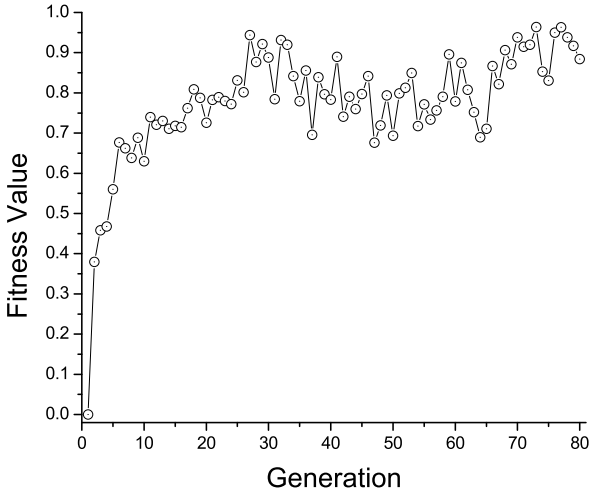

Figure 9.3: The average maximum fitness function of the neural network found at each generation in a GraspIt! environment.

9.2.5 Transitioning from Simulated to Physical Controller

The training method described in Section 9.2.3 is primarily done in simulation. However, transferring controllers evolved in simulation to the physical robot is challenging [40, 53, 57]. The main reason is that it is difficult to simulate physical properties such as friction and sensor and actuator characteristics with high enough fidelity to reproduce the simulated behaviors on real robots. In order to address this issue, we choose the following methods to improve the results of transfer to the real robot.

First, if the simulator is accurate enough, controllers that transfer well can be created simply by evolving them to be robust. That is, if sensor values and actuator responses frequently vary in simulation, the resulting controllers will be robust against small discrepancies between simulation and reality as well [28, 41, 59, 88]. Such uncertainties can be introduced into the simulation simply as noise, and solutions evolve that do not depend on accurate values and outcomes, thus transferring well.

If there are more systematic flaws in the simulation, behaviors may evolve that ex-

ploit them, and therefore transfer poorly. Such behaviors can be discouraged by incorporating transfer into evolution explicitly, by utilizing a multi-objective evolutionary algorithm that optimizes both a task-dependent controller fitness as well as a measure of how well the controller transfers from simulation to reality [45]. In any given generation, this method chooses at most one controller based on behavioral diversity to be evaluated on the real robot, requiring only a small number of hardware evaluations.

Another approach is to perform experiments on the real robot in order to improve the simulator, typically in one of two ways: (1) Experiments are performed on the real robot before running evolution to collect samples of the real world by recording sensor activations [59, 64]. When controllers are evaluated later during evolution, these samples are utilized to set the simulated sensor activations accurately. (2) Experiments are performed on the real robot during evolution to co-evolve the simulator and the controller, making an initially crude simulation more and more accurate [19, 99].

We will adopt the system-level simplex architecture [16] to provide safety guarantees during the transitioning. In this architecture, we use a simple and verified safety controller to ensure the stability and safety of the robot operations. This conservative safety control core is then complemented by a high-performance complex controller, which will be used whenever possible, but switch to the safety controller when system integrity is jeopardized.

9.3 Supporting Remote, Reliable and Real-Time Avatar-Human Communication

A key component of cyberphysical avatar technology is the remote, reliable and real-time avatar-human communication. Because of the mobility requirement of avatars to work in unstructured environments as is often the case in disaster recovery, wireless connection has to be established at the edge of the communication infrastructure. Owing to limited

wireless communication range (e.g. around 200m in Wi-Fi and 40m in 802.15.4-based [9] low-power wireless standards), a multi-hop wireless network in mesh topology is required to cover a large area, and more importantly, can overcome transmission blocking by objects such as metal doors in industrial facilities where avatars operate.

We envision the use of a combination of multi-hop wireless mesh networks and the existing Internet to support real-time communication between the avatars and human supervisor. However, the need for reliable and real-time communication imposes several significant challenges. Specifically, the current Internet architecture cannot guarantee any end-to-end QoS requirement for time-critical control flows and most existing wireless standards and routing protocols are not designed with real-time delay constraint in mind and as a result cannot provide any bound on end-to-end delay. Moreover, the inherent lossy wireless medium, the constantly fluctuating traffic volumes and channel conditions, together with complicated interference relationship have made it challenging to achieve high end-to-end reliability, which is essential for remote avatar-human communication.

A cyberphysical avatar typically contains two types of data flows. There is one or multiple data flows destinated to remote human supervisor containing physical information of the environment in which the avatar operates. These data flows include image flows captured by the cameras installed either on the avatar or in the operation environment, and real-time position/direction information of the avatar. There is also a control flow originated from the human operator to supervise the robot to execute designated tasks. Figure 9.4 presents an overview of our remote, reliable and real-time (R^3) communication infrastructure. We use Wi-Fi connection to transmit data flows because they require larger bandwidth but have soft real-time requirements on packet delivery. On the other hand, the time-critical control flow is transmitted on WirelessHART [81] real-time mesh network which is set up at the edge of the communication infrastructure to guarantee the end-to-end delay in the avatar-human communication. OpenFlow [58] network is deployed to enhance the existing Internet architecture to provide guaranteed QoS support. Our communication infrastructure

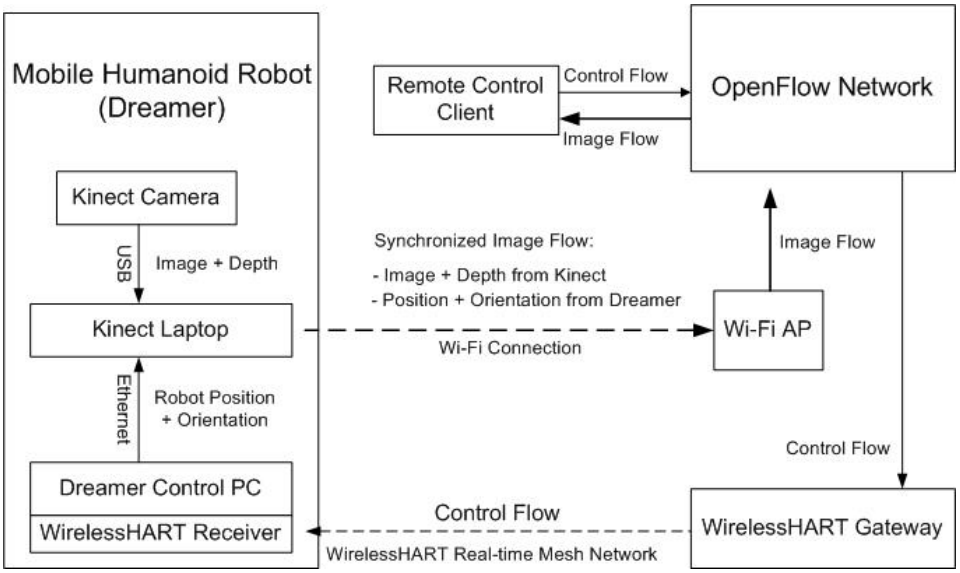

Figure 9.4: The R^3 communication infrastructure for cyberphysical avatars

has the following three key components.

9.3.1 Wi-Fi Connection for Supporting Data Flows

We use Wi-Fi connection at the edge of the communication infrastructure to help forward data flows to the remote human supervisor. In our current setting, the data flows are image streams captured by the Kinect sensor installed on the avatar, and the IP camera installed in its operation environment. Since the control PC on the avatar is battery-powered and its computation workload is intensive, the Kinect sensor is attached to a separated Laptop (Kinect Laptop). The Kinect Laptop and the avatar control PC are connected through Ethernet.

We utilize open source software library OpenKinect [12] to control the Kinect sensor. Both the color images and depth images received from the Kinect sensor will be synchronized with the position and orientation information received from the avatar control PC and sent to the remote supervisor. The remote control application receives the images and renders them on the user interface. It allows the operator to monitor the physical environ-

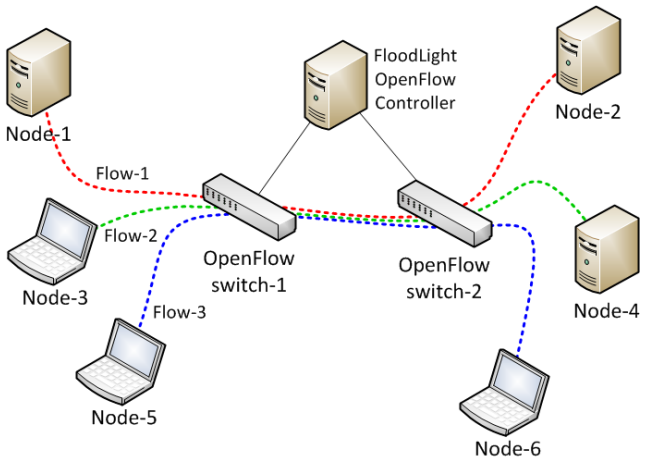

Figure 9.5: An overview of the OpenFlow testbed

ment the avatar is operating in and supervise it to execute designated tasks.

9.3.2 OpenFlow Network for Providing QoS Guarantees

Cyberphysical avatar technology is a highly interactive application that requires strict timing constraint on the control flows and guaranteed bandwidth for the data flows. Building such communication infrastructure is challenging because the current Internet architecture has no facility to guarantee minimum bandwidth and end-to-end latency for network flows. To address this problem, we enhance the existing Internet architecture by deploying OpenFlow [58] network to connect the remote human supervisor and the avatar operation environment.

To achieve end-to-end latency requirement for the control flow, we are enhancing OpenFlow switch to support Virtual Clock Server [90]. By using Virtual Clock Server, we can bound the queuing time that a packet goes through a switch. If we can bound the delay of each switch, then we can calculate and provide the end-to-end delay bound [29] from human controller to our avatar.

In order to demonstrate the minimum bandwidth guarantee provided by OpenFlow switches, we setup a testbed as shown in Fig. 9.5. In the testbed, we deploy two switches,

switch-1 (Pronto 3290) and switch-2 (Pronto 3295), and we enhance them with Indigo [10] open source OpenFlow firmware implementation. An OpenFlow controller PC running FloodLight OpenFlow controller [5] is used to configure the OpenFlow switches. We connect three PCs/Laptops to switch-1 to serve as clients, and three other PCs/Laptops are connected to switch-2 serving as servers. We run iperf to generate TCP flows with maximum rate. Each computer in this experiment is equipped with gigabit Ethernet network interface, and the maximum achievable throughput of each flow is around 900Mbps if no other traffic is present in this network.

We configure three flows in the testbed and evaluate their throughput with and without rate guarantee. In the baseline experiment without rate guarantee, we first start flow-1 at time 0, and we run flow-1 for 120 seconds. Flow-2 is started at time 30, and its duration is 60 seconds. Flow-3 starts at time 60 and runs for 30 seconds. In the second experiment with rate guarantee, we reserve 500Mbps bandwidth for flow-1 by setting a minimum rate guarantee queue at the egress port of switch-1. We then run the same setting as the baseline experiment.

Fig. 9.6 summarizes our experiment results. Fig. 9.6(a) shows the throughput of the three flows in the baseline experiment. Without rate guarantee, all three flows compete the link capacity of the connection link between switch-1 and switch-2. When three flows are present at the same time (from time 60 to 90), the throughput of each flow is around 300Mbps. In the second experiment as shown in Fig. 9.6(b), because we enable the minimum rate guarantee queue for flow-1, the throughput of flow-1 can be maintained around 500Mbps, even though in present of the other two flows.

9.3.3 IP-enabled WirelessHART Mesh for Supporting Control Flow

The control flow from the remote human supervisor to the robot control PC is time-critical and has hard deadline on its delivery. Due to the pervasive Wi-Fi signals and the backoff mechanism used in 802.11, the jitter in Wi-Fi transmission is large and unpredictable. This

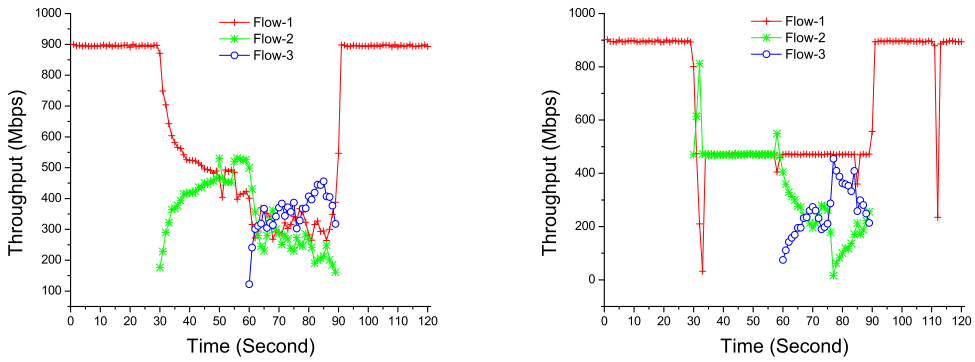

Figure 9.6: Demonstration of bandwidth guarantee in OpenFlow switch: (a) flow throughput without rate guarantee (b) flow throughput with rate guarantee

is a fatal disadvantage of Wi-Fi to be adopted for providing reliable and real-time communication for the control data flow in cyberphysical avatars.

For this reason, in the R^3 communication infrastructure designed for cyberphysical avatars, we use WirelessHART real-time mesh network for transmitting the control data flow to the avatar control PC. The control flow is first transmitted to the WirelessHART Gateway which is connected to the remote control application by OpenFlow network and then further relayed to the avatar control PC through WirelessHART mesh. We apply the IP-enabled communication stack and Gateway in this infrastructure, thus only the remote control application and the application running on avatar control PC need to understand the specific application protocol. The Gateway can remain unchanged when new services are established between the supervisor and the robot.

To demonstrate the benefit of WirelessHART real-time protocol, we conduct a set of experiments to compare the packets inter-arrival time (IAT) between Wi-Fi and WirelessHART. Both the Wi-Fi and WirelessHART network are deployed in the graduate student office at UT ACES 5th floor. We configure both the Wi-Fi and WirelessHART devices to periodically publish data every $20ms$ in the media access control (MAC) layer, and we calculate the IAT on the receiver side. For each network, we run the experiment independently

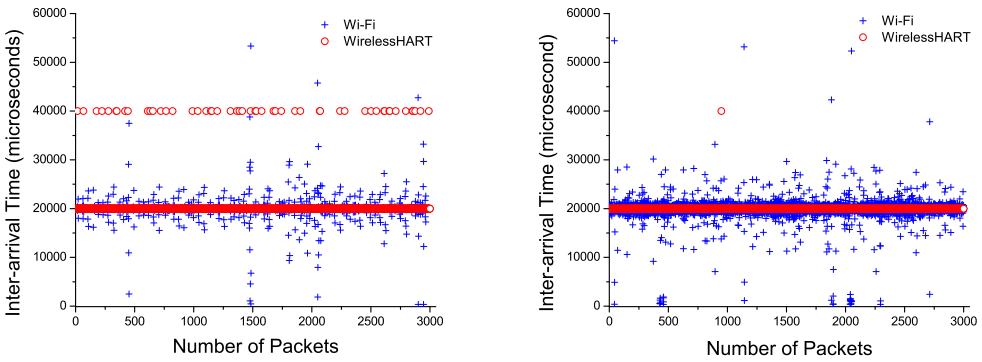

Figure 9.7: Inter-arrival time comparison between Wi-Fi and WirelessHART in office environment: (a) without present of jammer (b) in present of jammer

for 60 seconds and our results are summarized in Fig. 9.7.

Fig. 9.7(a) shows the IAT comparison in regular office environment. Because Wi-Fi utilizes CSMA-CA and random backoff mechanisms to coordinate channel access, it cannot transmit packets in a deterministic way, thus has high variation at the packet transmission time. WirelessHART on the other hand adopts TDMA mechanism and only transmits packets at fixed time points periodically. We observed from Fig. 9.7(a) that most WirelessHART packets are transmitted exactly every 20ms and only 1.7% of WirelessHART packets are retransmitted once due to interference from other Wi-Fi traffic, and have doubled IAT.

Fig. 9.7(b) shows the behavior of WirelessHART and Wi-Fi networks in present of intended interference. The jamming signal is created by deploying another Wi-Fi network, where we disable the carrier sense and random back mechanisms of its sender and use iperf to create maximum Wi-Fi traffic. Because of the interference from the jamming signal, we observed higher IAT variation in the Wi-Fi network. In WirelessHART network, we enable the dynamic channel blacklisting mechanism to mask out ill-conditioned channel. By hopping to less interference channel, most of WirelessHART packets are successfully transmitted without retransmission.

9.4 Designing and Building a Cyberphysical Avatar

We have completed a prototype of the cyberphysical avatar to verify the effectiveness of the proposed architecture. The remote control application is installed in UT ACES building and the robotic system is located in UT Human Centered Robotics Lab. OpenFlow switches are being deployed in UT campus network to provide bandwidth guarantee for reliable and real-time avatar-human communication. In this section, we will present the details of the system design and integration. A video demo of remote supervision on the avatar to execute "touch" and "incremental move" commands is available online [4].

9.4.1 System Integration in Human Centered Robotics Lab

Figure 9.8 presents an overview of the system setup in the Human Centered Robotics Lab. The control flow originated from the remote operator goes through the OpenFlow campus network and is transmitted to the Dreamer robot through the local real-time wireless communication subsystem. The Kinect camera and IP camera installed in the lab keep track of the robot's motion and its operation environment by sending image flows back to the remote supervisor. Based on these image flows, the remote human operator can supervise the robot to execute designated tasks by sending appropriate commands. Our system has the following three key components.

The Dreamer/Meka Hardware: The main hardware tool that we use for this study is the Dreamer/Meka mobile dexterous humanoid robot. This robot includes the T2 Meka torso, the A2 Series Elastic Meka arm, the H2 tendon driven Meka hand, the Dreamer/Meka head co-developed by Meka and UT Austin, and the torque-controlled holonomic UT Austin's Tricky base. The actuators for the base and the upper body, except for the head, contain torque/force sensors that enable Elmo amplifiers to implement current or torque feedback. An Ethercat serial bus communicates with sensors and motor amplifiers from a single computer system. A PC running Ubuntu Linux with the RTAI Realtime Kernel runs the models and control infrastructure described in Section 9.1. The Tricky holonomic base contains

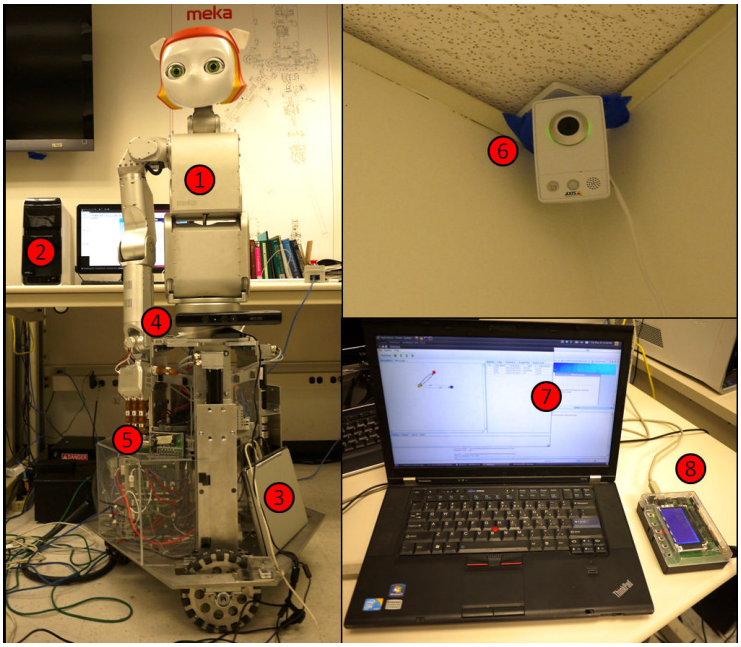

Figure 9.8: An overview of the system setup in UT Human Centered Robotics Lab. (1) Dreamer robot. (2) Robot control PC. (3) Kinect Laptop. (4) Kinect sensor. (5) WirelessHART receiver. (6) IP camera. (7) WirelessHART Gateway. (8) WirelessHART Access Point.

torque sensors as well as the inertial measurement unit (IMU) 3DM-GX3-25 from MicroStrain. It achieves holonomic motion and force capabilities by utilizing Omni wheels located in a equilateral triangular fashion.

Kinect/IP Cameras: We have two cameras installed in the Human Centered Robotics Lab. An IP camera is installed at the right upper corner to give an overview of the operation environment of the Dreamer robot; A Kinect camera is installed in front of the robot to capture the image and depth information of the target. As we mentioned in Section 9.3, due to the limitation on the power and the computation capability, the Kinect camera is installed on a separate Laptop (Kinect Laptop in Figure 9.8). The Kinect Laptop is connected to the avatar controller through Ethernet. It synchronizes image streams (including the image and depth information) captured from the Kinect camera and the position/orientation information of the robot together and sends to the remote control application.

Real-time wireless communication subsystem: We set up a WirelessHART network for achieving real-time wireless communication at the edge of the communication infrastructure. The communication subsystem includes a WirelessHART Gateway which is connect-

ed to the UT campus network, and a WirelessHART receiver which is connected to the avatar control PC to provide real-time communication. More intermediate devices can be deployed to form a mesh to cover larger area if necessary. The WirelessHART receiver exchanges control commands and robot status with the avatar control PC through shared memory. The robot status information is transmitted back to the remote operator on the WirelessHART real-time wireless network in the reversed direction. To provide deterministic communication, we establish a superframe with the size of 10 timeslots and create 5 pairs of transmit/receive links between the WirelessHART receiver and Gateway, so the frequency of the control flow can be up to 50Hz, which is sufficient for high-level control command transmission. As the ongoing work, we are enhancing the 802.11 standard with real-time and reliable features. We target at building a general wireless platform to support a wide range of wireless sensing and control applications by achieving a good balance among sampling rate, reliability, and energy efficiency.

9.4.2 Remote Control Application

Fig. 9.9 shows a screen capture of the remote control application we developed for supervising the Dreamer robot. The interface message between the Kinect Laptop and the remote client has two types, type A for color image data and type B for depth image data. For color image, the payload is an array of RGB data of the image. Each pixel is represented by 3 bytes (i.e. R, G, B in sequence). The image is scanned row by row, from up to bottom. In each row, it is scanned from left to right; For depth image, the payload is an array of the depth of the images. There is a depth value (2 bytes) for each pixel. The image is scanned row by row, from up to bottom. In each row, it is scanned from left to right. The interface message between the Dreamer robot and the remote client now has three command types, Move, Tough and Default. The payload of the message contains the x,y,z coordinates of the target. More command types will be added when the Dreamer robot is enhanced with more skills.

The specific skill we are training the Dreamer robot is to move itself close to a desk and pick up a designated target under supervision. As shown in Fig. 9.9, in the remote control UI, the color images and the depth images from Kinect camera and the images from the IP camera are displayed in the three image panels at the top of the UI. The human supervisor can choose a target in the color image from Kinect panel by clicking on it. After clicking on the target, a copy of the color image at this moment is copied to the image of target object panel. A red dot is added on the image showing the position the user clicked. The coordinate of this position is also displayed in the mouse clicked position label. Using the position of the click on the image and the depth data at that point, we calculate the physical coordinate of the target with respect to the Kinect. Once get the physical coordinate of the target, user can issue commands to the Dreamer robot to execute specific tasks. In our current testbed, two commands have been implemented already: "Default" and "Touch". The Default command gets the robot back to the default gesture, while the "Touch" command asks the robot to touch the target we clicked on. Fig. 9.10 is a sequence of video snapshots that demonstrate an user issued "Touch" commands to the robot. In the first video frame, the robot hand started from the default position. The robot moved to the commanded position as shown in frame 2 and 3. It is possible that the Kinect sensor has small measurement error due to its hardware limitation. In that case, we relied on visual feedback, and used incremental move commands to direct the robot to touch the target object as shown in frame 4. The grasp skill to lift up the target is under training and the Grasp command will be added to the remote control application as soon as the training is finished.

9.5 Summary

This chapter introduces the concept of a cyberphysical avatar which is defined to be a semi-autonomous robotic system that adjusts to an unstructured environment and performs physical tasks subject to critical timing constraints while under human supervision. A

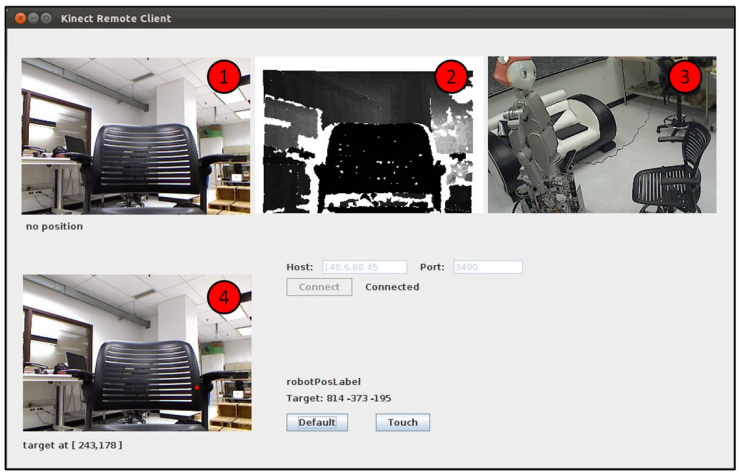

Figure 9.9: A screen capture of the remote control application for supervising the Dreamer robot. (1)(2) Color and depth image from Kinect sensor. (3) Image from IP camera. (4) Image snapshot when user presses the color image.

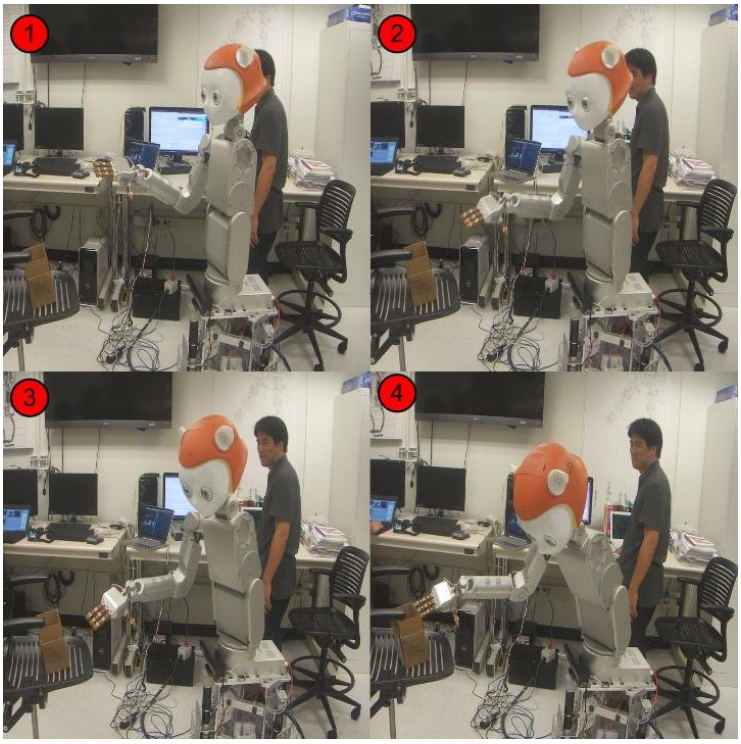

Figure 9.10: Video snapshots for demonstrating the "Touch" command.

cyberphysical avatar is the bridge technology that will help transition dumb teleoperated robotic devices to autonomous robots capable of functioning in unstructured environments. What makes the cyberphysical avatar possible today is the convergence of three recent technologies: body-compliant control in robotics, neuroevolution in machine learning and QoS guarantees in real-time communication. Body-compliant control is essential for safety since cyberphysical avatars will perform cooperative tasks in close proximity to humans. Neuroevolution technique is essential for "programming" cyberphysical avatars if they are to be used by non-experts for a large array of tasks, some unforeseen, in an unstructured environment. QoS-guaranteed real-time communication is essential to provide predictable, bounded-time response in human-avatar interaction. By integrating these technologies, we have built a prototype cyberphysical avatar testbed. Building this physical testbed is an essential first step for exploring the cyberphysical avatar concept because the physical and computational complexities involved necessarily require less than exact modeling and the use of mathematical approximations to simulate physical processes; the viability of the cyberphysical avatar must be validated by a physical testbed. As of this time, the basic body-compliant controller and the communication subsystems have been integrated; work is ongoing to improve the interface between the robot and the controller and to train the robot by the NEAT algorithm.

Chapter 10

Conclusion

10.1 Summary

Cyber-physical systems consist of the class of large-scale infrastructures that have significant cyber and physical components and have wide-ranging impact on society in their deployment. This thesis aims at providing a foundation for designing networking infrastructure and data management techniques to help build large-scale reliable and secure cyber-physical systems supporting a wide range of time-critical services. We first present a TDMA-based low-power and secure real-time wireless protocol called WirelessHART, which can serve as an ideal communication infrastructure for CPS subsystems. We describe the network management techniques designed for ensuring the reliable routing and real-time services inside the subsystems and data management techniques for maintaining the quality of the sampled data from the physical world. To interconnect heterogeneous CPS subsystems together, we further enhance the WirelessHART protocol with an IP adaptation layer and a constrained application layer. This enhancement makes the underlying connectivity technologies transparent to the application developers thus enables rapid application development, incremental deployment and efficient migration among different CPS platforms. We are applying the proposed networking infrastructure and data management techniques

to many cyber-physical applications where remote, reliable and real-time communication and data processing are required. These applications include the remote real-time welding system, the network-based mobile gait rehabilitation system and the cyberphysical avatar.

The following section summarizes several future research topics.

10.2 Future Research

10.2.1 An Adjustable High-speed Real-time and Reliable Wireless Platform

We are working on further enhancing the WirelessHART protocol to serve as a general platform for CPS subsystems by supporting a wider range of real-time and reliable applications including high-speed wireless control systems. The following summarizes our ongoing works and the challenges to tackle.

Adjustable TDMA State Machine for Supporting Dynamic Traffic

The timeslot in WirelessHART data link layer is fixed at 10ms by default. This restricts our current platform can only be applied to applications whose required data sampling rate is no larger than 100Hz. With the 802.15.4 PHY layer unchanged, to support a wider range of applications, we are working towards extending the current data link layer to make the size of the timeslot adjustable. There are several challenges to be addressed in this enhancement: 1) How to make a proper compromise between the timeslot size and the corresponding maximum frame size to be supported and the security mechanism to be applied; 2) How to adjust the TDMA state machine online to support dynamic traffic while still maintaining network-wide synchronization.

Dual-CCA Technique for Improving Co-existence Performance

The built-in channel hopping mechanism in WirelessHART diversifies the channel usage among all the active channels and significantly improves the communication latency com-

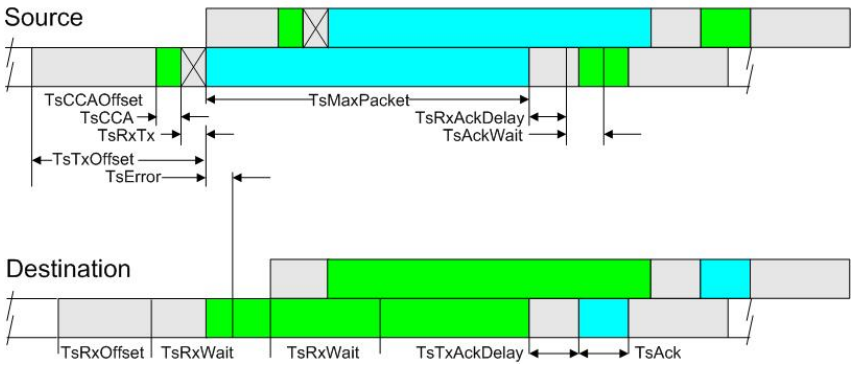

Figure 10.1: Timing of dual CCA technique

pared with the channel-fixed solution. In WirelessHART, a CCA will be conducted in each communication transaction before the packet is transmitted. If the channel is noisy, we will give up the transmission and wait for the next available timeslot. Since the CCA time is small (128 μs), as shown in Figure 10.1, we are proposing the dual-CCA technique to further improve the co-existence performance. The dual-CCA technique conducts two CCAs on different channels in the same timeslot. If the primary channel is not clear, the sender will check the secondary channel instead, and transmit the packet on the second channel if it is clear. Only if both channels are noisy, we will wait for the next available timeslot for transmission. There are several issues to take care in designing the dual-CCA protocol: 1) The data link layer state machines on both the sender and receiver sides need to be modified to synchronize on sending/receiving packets on different channels; 2) The primary and secondary channels in the same timeslot should be separated far enough to avoid interference from Wi-Fi; and 3) A scheduling algorithm should be carefully designed to achieve global confliction-free channel assignment.

Real-time Wi-Fi for Supporting High-speed Wireless Control Applications

Due to their enhanced mobility and reduced configuration and maintenance cost, wireless control systems have been employed in many sensing and control applications. Howev-

er, most existing wireless control systems focus on monitoring and low speed control, and less effort has been made on high speed wireless control applications. It is because most existing wireless communication protocols cannot provide real-time and reliable communication links with preferable high speed ($\geq$ 1000Hz) by taking energy saving into consideration. Even with the enhancement of the adjustable TDMA state machine, our WirelessHART communication protocol can only reduce the time slot length to 3 milliseconds, thus supporting up to 333Hz sampling rate. This still cannot satisfy the requirement of many high-speed wireless control applications.

To address this problem, we propose Real-Time WiFi (RT-WiFi) which is a high-speed real-time and reliable wireless protocol combining the advantages of WirelessHART and Wi-Fi together. At the very bottom, RT-WiFi adopts physical layer of Wi-Fi in order to support high data rate. On top of that, we are hacking the MAC layer of Wi-Fi to adopt TDMA for providing real-time data delivery and to explore the channel diversity. The length of each time slot in RT-WiFi MAC layer is set as 500 microseconds, so that we can achieve a data rate of 2kHz which is sufficient for a wide range of wireless control applications. In order to provide reliable communication, RT-WiFi utilizes channel hopping and channel blacklists mechanisms to avoid interference. It also takes an energy-efficient design by turning on its wireless radio only in time slots when transmitting or receiving is scheduled, and it aggressively puts devices in power saving mode to minimize energy consumption. We envision that RT-WiFi can serve as an ideal platform to high speed wireless control systems. By adjusting the data rate of RT-WiFi, our wireless platform can support a wide range of wireless control applications and achieve good balance among sampling rate, reliability, and energy efficiency.

10.2.2 Unanswered Questions in Data Management Techniques

In Chapter 6, we have shown that both of *DS-FP* and *DS-LALF* can significantly reduce the CPU workload incurred by the update transactions in the system. However, there are

still many unanswered questions about *DS-FP* and *DS-LALF*. First of all, in Chapter 6, we have shown through counterexamples that although *DS-FP* and *DS-LALF* can significantly reduce the CPU workload, neither of them is optimal in terms of schedulability. An important question would be what is an optimal scheduling algorithm for maintaining data freshness? Second, the proposed pattern search and schedulability test algorithms for both *DS-FP* and *DS-LALF* are computation intensive. Can we find better necessary and sufficient conditions for *DS-FP* and *DS-LALF* to improve their time and space complexity? Third, our current study on real-time data freshness maintenance is restricted on single CPU platform and only consider independent update transaction set. Future research is needed for how well they will work on multi-core platforms and for dependent task sets.

To continue our research work in Chapter 7 on maintaining data freshness in dynamic cyber-physical systems, and study the properties of the scheduling switch for wider classes of scheduling policies, there are also several open questions to be answered: 1) Suppose the MCR latency requirement is infinite, if the old and new task sets are schedulable under the old and new scheduling policies respectively, does there exist a proper switch point using SBS or ABS? 2) SBS and ABS are both synchronous algorithms and all the tasks in the new mode are released simultaneously. Can we design asynchronous algorithms? If so, how should scheduling switch be conducted?

Another important future work is to study the performance of the proposed data management techniques in practical Cyber-Physical Sensing and Control Systems such as the network-based mobile gait rehabilitation system and the cyberphysical avatar.

10.2.3 New Avenues in the Cyberphysical Avatar Technique

The availability of the cyberphysical avatar testbed opens up new avenues for future research that arise from the interaction between robotics, machine learning and real-time system design. We mention below a couple of the issues to be explored.

- Models for real-time resource allocation and scheduling must be explored that are

neither hard nor soft real-time in that a cyberphysical avatar is best characterized as a hierarchical, ”contract-driven” real-time system. For example, the avatar may be commanded by the human supervisor to trade off walking speed against visual processing accuracy of its environment; in an uncluttered environment, the avatar can move faster without worrying about being tripped over by an unexpected obstacle. This might involve trading off the quality of the sensor data for faster response time in the gait-maintenance control loop, while keeping the related mode change overhead low. This calls for a proactive approach in providing performance guarantees.

- Concepts of compositionality may be introduced that will make it easier to compose physical components into higher-level semantic units. For example, in compliant control, postures are objectives that need to optimize a performance objective in the null space of tasks (which are coordinates to track a position or force trajectory). Since attaining desired postures requires non-trivial computation in real time, fast tests for checking feasibility will be very useful for helping the human supervisor to make decisions. This might be possible if we can lift the level of abstraction of the compliance problem to a mixed logical-numerical constraint satisfaction formulation and exploit techniques in run-time verification.

Bibliography

[1] 1321xEVK Product Summary. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=1322x_Dev_Kits.

[2] Bluetooth. www.bluetooth.com/.

[3] Constrained Application Protocol (CoAP). http://datatracker.ietf.org/doc/draft-ietf-core-coap/.

[4] Cyberphysical Avatar Demo. http://www.youtube.com/watch?feature=player_embedded&v=hyeWLAgsM9k.

[5] Floodlight openflow controller. http://floodlight.openflowhub.org/.

[6] Freescale. www.freescale.com.

[7] GraspIt! http://sourceforge.net/projects/graspit/files/releases/.

[8] IEEE 802.11 Task Group. grouper.ieee.org/groups/802/11.

[9] IEEE 802.15.4 WPAN Task Group. www.ieee802.org/15/pub/TG4.html.

[10] Indigo open source firmware for openflow swiches. http://www.openflowhub.org/display/Indigo/.

[11] Java Universal Network/Graph Framework. jung.sourceforge.net/.

[12] OpenKinect. www.openkinect.org.

[13] ZigBee Alliance. http://www.zigbee.org.

[14] M. Amirijoo, J. Hansson, S. H. Son, and S. Member. Specification and management of QoS in real-time databases supporting imprecise computations. *IEEE Transactions on Computers*, pages 304–319, 2006.

[15] P. Bahl, R. Chandra, and J. Dunagan. SSCH: Slotted seeded channel hopping for capacity improvement in ieee 802.11 ad-hoc wireless networks. In *Proceedings of ACM International Conference on Mobile Computing and Networking*, 2004.

[16] S. Bak, D. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha. The system-level simplex architecture for improved real-time embedded system safety. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 99 –107, april 2009.

[17] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of IEEE Real-Time Systems Symposium*, 1990.

[18] M. H. Bateni, L. Golab, M. T. Hajiaghayi, and H. Karloff. Scheduling to minimize staleness and stretch in real-time data warehouses. In *Proceedings of the Annual Symposium on Parallelism in Algorithms and Architectures*, 2009.

[19] J. C. Bongard and H. Lipson. Once more unto the breach: Co-evolving a robot and its simulator. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*, pages 57–62. MIT Press, 2004.

[20] A. Cardenas, S. Amin, and S. Sastry. Secure control: Towards survivable cyber-physical systems. In *Proceedings of IEEE International Conference on Distributed Computing Systems*, 2008.

[21] K. don Kang, S. H. Son, J. A. Stankovic, and T. F. Abdelzaher. A qos-sensitive approach for timeliness and freshness guarantees in real-time databases. In *Prceedings of Euromicro Conference on Real-Time Systems*, 2002.

[22] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*. NIST Special Publication 800-38C, May 2004.

[23] P. Emberson and I. Bate. Minimising task migration and priority changes in mode transitions. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2007.

[24] G. Fiore, V. Ercoli, A. J. Isaksson, K. Landernäs, and M. D. Di Benedetto. Multihop multi-channel scheduling for wireless control in WirelessHART networks. In *Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation*, 2009.

[25] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: From architectures to learning. *Evolutionary Intelligence*, 1:47–62, 2008.

[26] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(4), 2001.

[27] L. Golab, T. Johnson, and V. Shkapenyuk. Scheduling updates in a real-time stream warehouse. In *Proceedings of IEEE International Conference on Data Engineering*, 2009.

[28] F. Gomez and R. Miikkulainen. Transfer of neuroevolved controllers in unstable domains. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2004.

[29] P. Goyal, S. Lam, and H. Vin. Determining end-to-end delay bounds in heterogeneous networks. *Network and Operating Systems Support for Digital Audio and Video*, 1018:273–284, 1995.

[30] R. P. Grimaldi. Discrete and combinatorial mathematics: An applied introduction. *Addison-Wesley*, 1998.

[31] T. Gustafsson and J. Hansson. Data management in real-time systems: a case of on-demand updates in vehicle control systems. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004.

[32] S. Han, D. Chen, M. Xiong, K.-Y. Lam, A. K. Mok, and K. Ramamritham. Schedulability analysis of deferrable scheduling algorithms for maintaining real-time data freshness. *UTCS Technical Report TR-11-38*, 2011.

[33] S. Han, D. Chen, M. Xiong, and A. Mok. A schedulability analysis of deferrable scheduling using patterns. In *Prceedings of Euromicro Conference on Real-Time Systems*, 2008.

[34] S. Han, A. K. Mok, J. Meng, Y.-H. Wei, X. Zhu, L. Sentis, K. S. Kim, R. Miikkulainen, and J. Menashe. Architecture of a cyberphysical avatar. *UTCS Technical Report TR-12-12*, 2012.

[35] S. Han, K. yiu Lam, J. Wang, K. Ramamritham, and A. K. Mok. On co-scheduling of update and control transactions in real-time sensing and control systems: Algorithms, analysis and performance. *IEEE Transactions on Knowledge and Data Engineering*, 2012.

[36] S. S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, Upper Saddle River, New Jersey, third edition, 2009.

[37] R. Henia and R. Ernst. Scenario aware analysis for complex event models and distributed systems. In *Proceedings of IEEE Real-Time Systems Symposium*, 2007.

[38] S.-J. Ho, T.-W. Kuo, and A. K. Mok. Similarity-based load adjustment for real-time data-intensive applications. In *Proceedings of IEEE Real-Time Systems Symposium*, 1997.

[39] M. Ilic, L. Xie, U. Khan, and J. Moura. Modeling future cyber-physical energy systems. In *IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*, 2008.

[40] N. Jakobi. *Minimal Simulations for Evolutionary Robotics*. PhD thesis, School of Cognitive and Computing Sciences, University of Sussex, 1998.

[41] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J. Merelo, and P. Chacon, editors, *Advances in Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 704–720. Springer, Berlin, 1995.

[42] K. D. Kang, J. Oh, and S. H. Son. Chronos: Feedback control of a real database system performance. In *Proceedings of IEEE Real-Time Systems Symposium*, 2007.

[43] K. D. Kang, S. H. Son, and J. A. Stankovic. Managing deadline miss ratio and sensor data freshness in real-time databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1200–1216, 2004.

[44] W. Kang, S. H. Son, and J. Stankovic. QeDB: A quality-aware embedded real-time database. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 108–117, 2009.

[45] S. Koos, J.-B. Mouret, and S. Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 119–126, 2010.

[46] D. Kulkarni, C. V. Ravishankar, and M. Cherniack. Real-time, load-adaptive processing of continuous queries over datastreams. In *Proceedings of the International Conference on Distributed Event-based Systems*, 2008.

[47] T.-W. Kuo and A. Mok. Ssp: A semantics-based protocol for real-time data access. In *Proceedings of IEEE Real-Time Systems Symposium*, 1993.

[48] A. Labrinidis and N. Roussopoulos. Update propagation strategies for improving the quality of dataon the web. In *Proceedings of International Conference on Very Large Data Bases*, 2001.

[49] K.-Y. Lam, M. Xiong, B. Y. Liang, and Y. Guo. Statistical quality of service guarantee for temporal consistency of real-time data objects. In *Proceedings of IEEE Real-Time Systems Symposium*, 2004.

[50] E. A. Lee. Cyber physical systems: Design challenges. In *Object-Oriented Real-Time Distributed Computing*, pages 363–369, 2008.

[51] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 201 –209, 1990.

[52] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 1982.

[53] H. Lipson, J. Bongard, V. Zykov, and E. Malone. Evolutionary robotics for legged machines: From simulation to physical reality. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, pages 11–18, 2006.

[54] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 1973.

[55] D. Locke. Real-time databases: Real-world requirements. In A. Bestavros, K. J. Lin,

and S. H. Son, editors, *Real-Time Database Systems: Issues and Applications*, pages 83–91. Kluwer Academic, 1997.

[56] M. K. Marina and S. R. Das. On-demand multipath distance vector routing in ad hoc networks. In *Proceedings of IEEE International Conference on Network Protocols*, 2001.

[57] M. Mataric and D. Cliff. Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19:67–83, 1996.

[58] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 2008.

[59] O. Miglino, H. H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial Life*, 2:417–434, 1995.

[60] R. Miikkulainen. Neuroevolution. In *Encyclopedia of Machine Learning*. 2010.

[61] S. A. Mohammed Tariquea, Kemal E. Tepeb and S. Erfanib. Survey of multipath routing protocols for mobile ad hoc networks. *Journal of Network and Computer Applications*, 32(6), 2009.

[62] S. Mueller, R. Tsang, and D. Ghosal. Multipath routing in mobile ad hoc networks: Issues and challenges. *Performance Tools and Applications to Networked Systems*, 2004.

[63] NIST. *Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197, November 2001.

[64] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In *Proceedings of the Fourth*

International Workshop on the Synthesis and Simulation of Living Systems, pages 190–197, 1994.

[65] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *Prceedings of Euromicro Conference on Real-Time Systems*, 1998.

[66] H. Qu and A. Labrinidis. Preference-aware query and update scheduling in web-databases. In *Proceedings of IEEE International Conference on Data Engineering*, pages 356–365, 2007.

[67] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-Physical Systems: The Next Computing Revolution. In *Proceedings of the Design Automation Conference*, 2010.

[68] K. Ramamritham. Real-time databases. *Distributed and Parallel Databases*, 1(2):199–226, 1993.

[69] K. Ramamritham. Where do time constraints come from and where do they go. *International Journal of Database Management*, 1996.

[70] K. Ramamritham, S. H. Son, and L. C. Dipippo. Real-time databases and data services. *Real-Time Systems*, 28(2):179–215, 2004.

[71] J. Real. Mode change protocols for real-time systems. *Ph.D. Thesis*.

[72] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 2004.

[73] A. Saifulah, C. Lu, Y. Xu, and Y. Chen. Real-time scheduling for WirelessHART networks. In *Proceedings of IEEE Real-Time Systems Symposium*, 2010.

[74] L. Sentis. *Motion Planning for Humanoid robots*, chapter Compliant Control of Whole-Body Multi-Contact Behaviors in Humanoid Robots, pages 29–63. Springer Berlin Heidelberg, 2010.

[75] L. Sentis, J. Park, and O. Khatib. Compliant control of multi-contact and center of mass behaviors in humanoid robots. *IEEE Transactions on Robotics*, 26(3):483–501, June 2010.

[76] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang. Cyber-physical systems: A new frontier. In *Proceedings of IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing*, pages 1–9, 2008.

[77] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1988.

[78] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[79] S.J.Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In *Proceedings of IEEE International Conference on Communications*, 2001.

[80] P. Soldati, H. Zhang, and M. Johansson. Deadline-constrained transmission scheduling and data evacuation in wirelesshart networks. *Technical Report TRITA-EE 2008:060*, 2008.

[81] J. Song, S. Han, A. K. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt. WirelessHART: Applying wireless technology in real-time industrial process control. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008.

[82] J. A. Stankovic, S. H. Son, and J. Hansson. Misconceptions about real-time databases. *IEEE Computer*, 32, 1999.

[83] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[84] N. Stoimenov, S. Perathoner, and L. Thiele. Reliable mode changes in real-time systems with fixed priority or edf scheduling. In *Proceedings of the Conference on Design, Automation, & Test in Europe*, 2009.

[85] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[86] M. Thiele, A. Bader, and W. Lehner. Multi-objective scheduling for real-time data warehouses. *Computer Science-Research and Development*, 24(3):137–151, 2009.

[87] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[88] V. Valsalam. *Utilizing Symmetry in Evolutionary Design*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2010. Technical Report AI-10-04.

[89] J. Xiang, G.-H. Li, H.-J. Xu, and X.-K. Du. Data Freshness Guarantee and Scheduling of Update Transactions in RTMDBS. In *Proceedigns of International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4, 2008.

[90] G. G. Xie and S. S. Lam. Delay guarantee of virtual clock server. *IEEE/ACM Transactions on Networking*, 1995.

[91] M. Xiong, S. Han, and D. Chen. Deferrable scheduling for temporal consistency: Schedulability analysis and overhead reduction. In *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2006.

[92] M. Xiong, S. Han, and K.-Y. Lam. A deferrable scheduling algorithm for real-time transactions maintaining data freshness. In *Proceedings of IEEE Real-Time Systems Symposium*, 2005.

[93] M. Xiong, S. Han, K.-Y. Lam, and D. Chen. Deferrable scheduling for maintaining real-time data freshness: Algorithms, analysis, and results. *IEEE Transactions on Computers*, 2008.

[94] M. Xiong and K. Ramamritham. Deriving deadlines and periods for real-time update transactions. In *Proceedings of IEEE Real-Time Systems Symposium*, 1999.

[95] M. Xiong and K. Ramamritham. Deriving deadlines and periods for real-time update transactions. *IEEE Transactions on Computers*, 53(5):567–583, 2004.

[96] M. Xiong, R. Sivasankaran, J. A. Stankovic, K. Ramamritham, and D. Towsley. Scheduling transactions with temporal constraints: Exploiting data semantics. *IEEE Transactions on Knowledge and Data Engineering*, 1996.

[97] M. Xiong, Q. Wang, and K. Ramamritham. On earliest deadline first scheduling for temporal consistency maintenance. *Real-Time Systems*, 2008.

[98] Z. Ye, S. Krishnamurthy, and S. Tripathi. A framework for reliable routing in mobile ad hoc networks. In *Proceedings of IEEE International Conference on Computer Communications*, 2003.

[99] J. C. Zagal and J. Ruiz-Del-Solar. Combining simulation and reality in evolutionary robotics. *Journal of Intelligent and Robotic Systems*, 50:19–39, 2007.

[100] H. Zhang, P. Soldati, and M. Johansson. Optimal link scheduling and channel assignment for convergecast in linear WirelessHART networks. *Technical Report TRITA-EE 2009:018*, 2009.